**Lab Notebook Table of Contents**

|  |  |
| --- | --- |
| Week 1: Week of June 5th | Pg. 2 |
| Week 2: Week of June 12th  Week 3: Week of June 19th  Week 4: Week of June 26th  Week 5: Week of July 3rd  Week 6: Week of July 10th  Week 7: Week of July 17th  Week 8: Week of July 24th  Week 9: Week of July 31st  Week 10: Week of August 7th | Pg. 10  Pg. 19  Pg. 62  Pg. 82  Pg. 89  Pg. 103  Pg. 110  Pg. 113 |
|  |  |
|  |  |
|  |  |

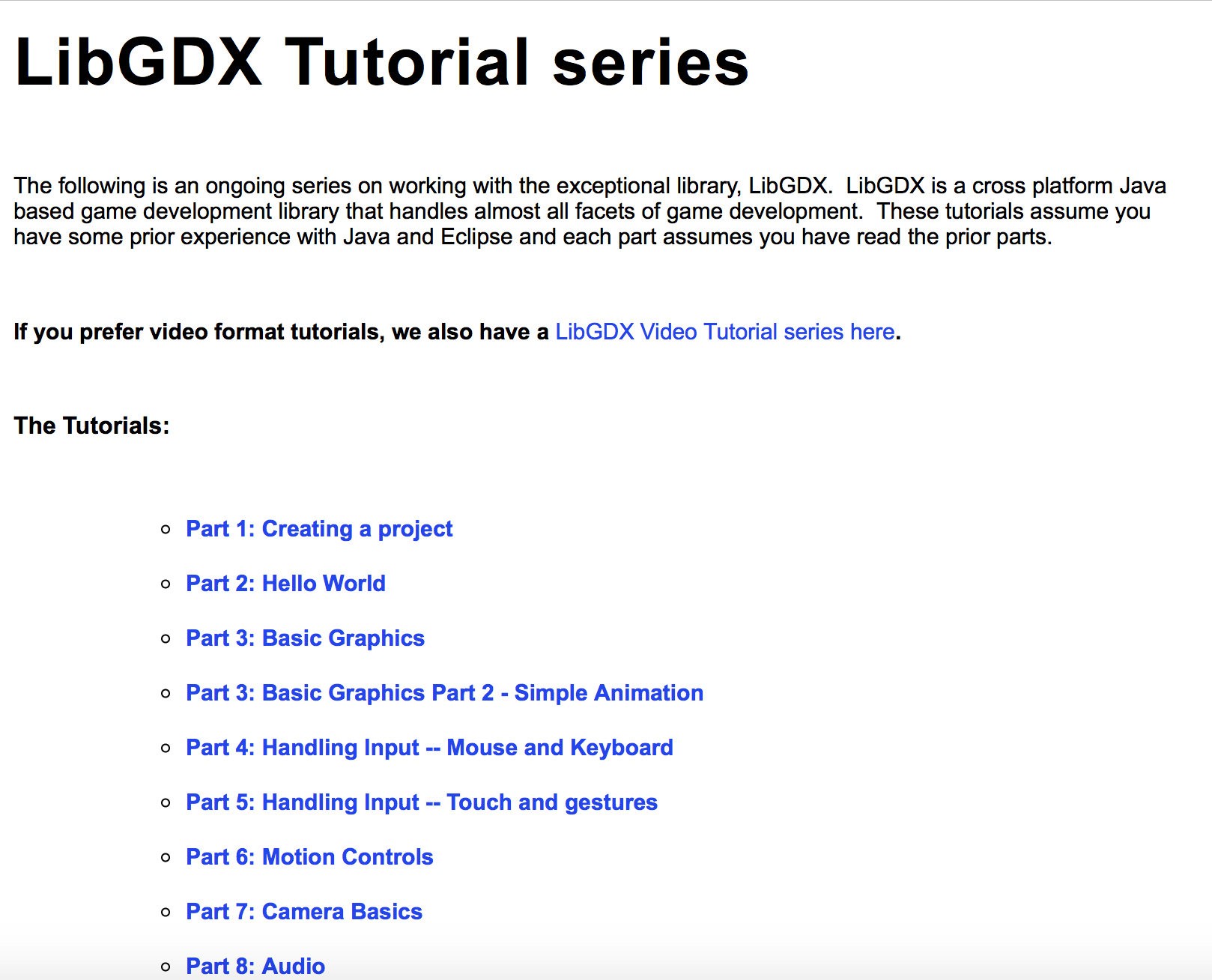
**Week 1: Week of June 5th**

*Day 1: June 5th:*

NYU Tandon Summer Undergraduate Research Program had its first general orientation where they introduced the students to each other and to the details/logistics of the program. The BMS specific orientation introduced students working on BMS/CBE projects to each other. Eugene and I read through the Chemtris research poster from last year and discussed what we would need to do in order to effectively work on the project this summer. Both of us have previous programming experience but neither of us have experience in the certain game design software used for this project (libGDX in Unity) so we will need to use online tutorials and courses to learn the software.

*Day 2: June 6th:*

We spent a couple hours as a group on the Python course tutorials to get the basic idea of Python using the Python updated software shell as well as downloaded Visual Studio in order to organize/color code the Python code and to actually run the code. I also started an online LibGDX Tutorial series to learn the basics of creating a project, making graphics, handling input motion controls, camera basics and audio

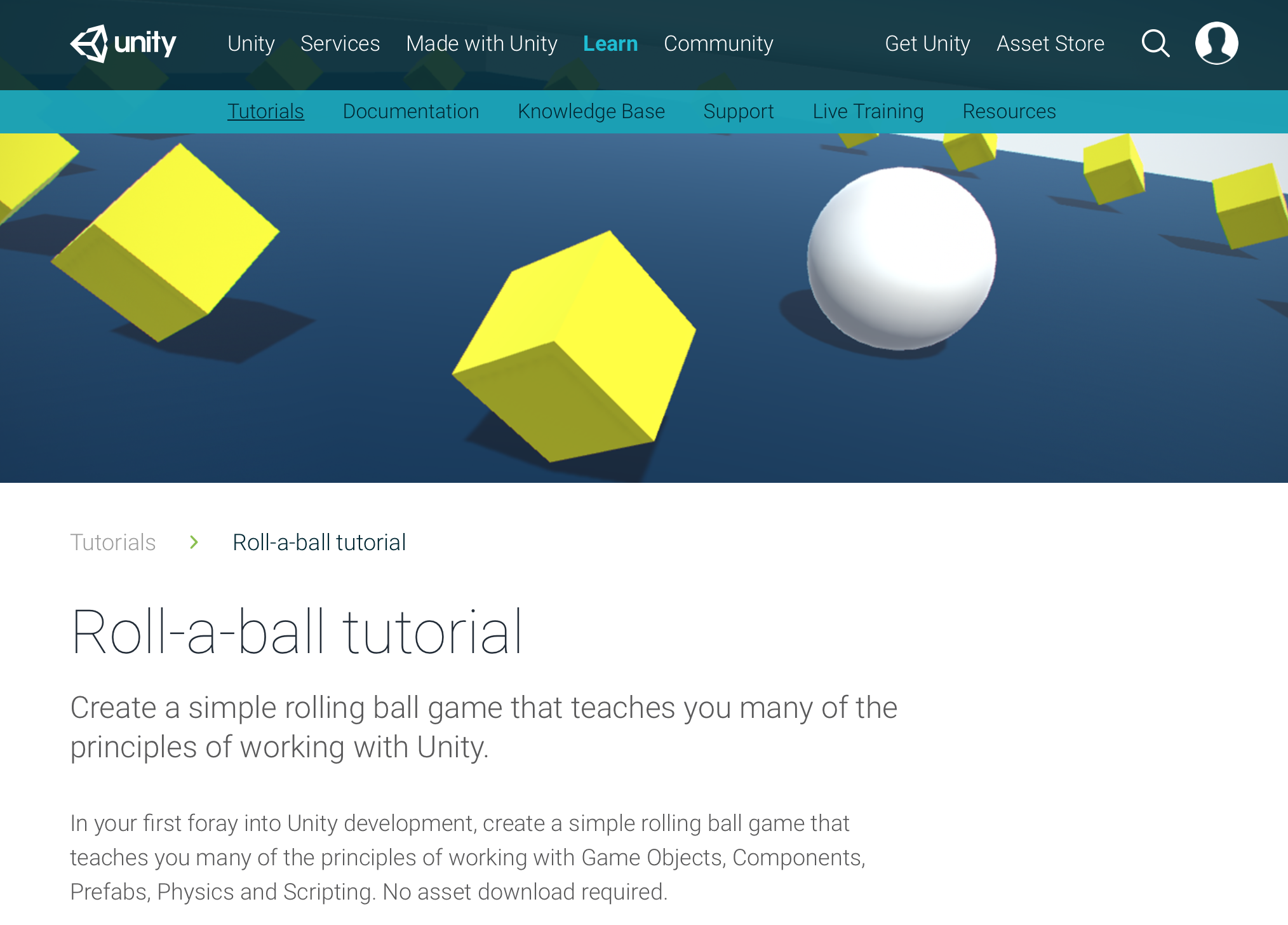


Link to tutorial (text): <http://www.gamefromscratch.com/page/LibGDX-Tutorial-series.aspx>

Link to tutorial (video): <http://www.gamefromscratch.com/page/LibGDX-Video-Tutorial-Series.aspx>

*Day 3: June 7th*

The Chemtris game is only available on the Android app store so I used Eugene’s phone to try and play it. The game as it is is kind of confusing and the icons and control buttons all loaded separately one by one instead of all at once, there was no music, the game crashed a couple of times after some of the levels (which are all problems that need to be fixed). We would ideally like to make it more interactive and interesting because the point of the game is to be educational and encourage students to want to learn about chemistry/organic chemistry. In order to be able to work and code on this project effectively, both of us needed to learn Unity and experiment with different colors and concepts. I started working on the tutorials on the official Unity website. I wasn’t able to download Unity due to problems with my computer before I did the first tutorial which was the Roll-a-ball tutorial so I used Eugene’s laptop to follow along.

**

This Roll-a-ball tutorial required me to create a game object, moving the player so that the player collects the levitating yellow collectible cubes, moving the camera and increasing the range of the play area, collecting the objects and initiating a displayable scoreboard, building the game and exporting it. It also incorporated making design choices on color, depth, shapes to make the game more interesting.

Moving the player that goes around and collects the objects to gain points required code using C#. This code was incorporated under the PlayerController component of the game object:

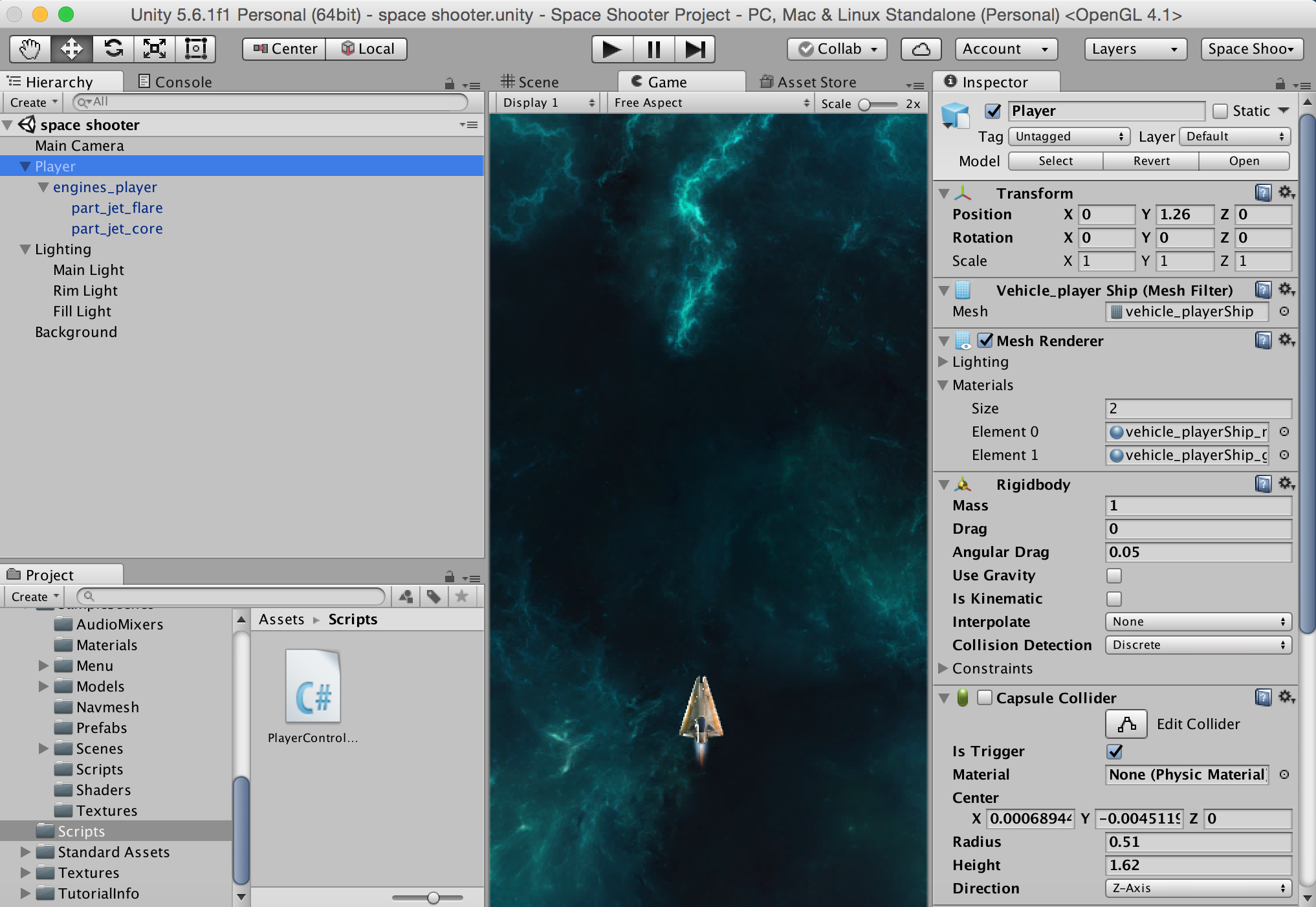
using UnityEngine;  
using System.Collections;  
  
public class PlayerController : MonoBehaviour {  
  
 public float speed;  
  
 private Rigidbody rb;  
  
 void Start ()  
 {  
 rb = GetComponent<Rigidbody>();  
 }  
  
 void FixedUpdate ()  
 {  
 float moveHorizontal = [Input.GetAxis](http://docs.unity3d.com/Documentation/ScriptReference/Input.GetAxis.html) ("Horizontal");  
 float moveVertical = [Input.GetAxis](http://docs.unity3d.com/Documentation/ScriptReference/Input.GetAxis.html) ("Vertical");  
  
 Vector3 movement = new Vector3 (moveHorizontal, 0.0f, moveVertical);  
  
 rb.AddForce (movement \* speed);  
 }  
}

After I finished the first tutorial, I went home and downloaded Unity, and started working on setting up the project for the second tutorial which was the Space Shooter project. I downloaded the Space Shooter Assets bundle which allowed me to use the specific spaceship, background space effects and music needed for the project.

*Day 4: June 8th:*

We went as a group on the MakerSpace tour to check out the 3D printers and safety/how to use them/how to get trained. The guide introduced us to the software used and how to download a project on the computer, change the dimensions, and save the project to the SD card which is then inserted into the 3D printer. We also learned how to set up the 3D printer using the settings and how to test the 1mm distance between the nozzle and the platform using a piece of paper. After training, we all got our IDs checked out. The MakeSpace tour gave me the idea that along with modifying the Chemtris game and making it adaptable to iPhone and potentially VR and having the game for display so people could play it during the poster presentations, it would also be cool if we could print out the molecules at a certain level to actually show the tangible 3D version of the molecules for a better understanding of what the goal of the game is.

I spent the next two hours continuing to work on the second installment of the Space Shooter tutorial. The second installment for the Space Shooter tutorial consisted of setting up the project, creating the player GameObject, moving the camera and modifying the lighting.



As seen in this, I created a player GameObject that has the engines\_player asset that has a part\_jet\_flare and part\_jet\_core elements which give the space ship the appearance of moving through space. The lighting was also divided into the main light, rim light, and fill light and the position, rotation, scale and intensity of all the lights were modified to give the appearance that light was targeting the space ship on one side. The background was also implemented by importing a space texture asset and modifying the light to make it more darker in contrast to the spaceship.

The Library Resource Presentation presented by NYU Bern Dibner Library was very helpful to find specific resources related to our projects. Using NYU Library databases for journal articles such as SciFinder and Elsevier were ideal when using keywords. Because Chemtris is a video game and there haven’t been many journal articles written about video games, it gave us the flexibility to do research about relevant topics such as organic chemistry, game design, educational video games and also allowed us to relate the game back to problems in medicine caused by an improper bonding between molecules. Professor Seidenstein also suggested researching if there has been a patent on the name/game Chemtris and if there isn’t a patent, recommended that we also research the process through which we can get a patent with our names on it (most likely through NYU). After the presentation, I continued to use the databases he introduced to us to research relevant articles to my project. I also researched potential resources that could help with creating the game such as the Brooklyn Experimental Media Center, CITE Game Innovation Lab, Games for Learning Institute, NYU Game Center and Video Lab and NYU Wireless. I also got in touch with Benson Kuang who put me in touch with Mohammed who worked on this project before. He was able to give me a link to the game (Chemtris Demo (Alpha Stage)): <https://www.youtube.com/watch?v=GAVTbFGHFJo&feature=youtu.be>

*Day 5: June 9th*

I continued working on the Space Shooter tutorial beginning with modifying the background a little bit more and writing some code to move the space shooter based on the user’s input:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerController : MonoBehaviour

{

public float speed;

private Rigidbody rb;

void Start ()

{

rb = GetComponent<Rigidbody> ();

}

void FixedUpdate (){

float moveHorizontal = Input.GetAxis ("Horizontal");

float moveVertical = Input.GetAxis ("Vertical");

Vector3 movement = new Vector3 (moveHorizontal, 0.0f, moveVertical);

rb.velocity = movement \* speed;

}

}

The code was later modified to be:

using UnityEngine;  
using System.Collections;  
  
[System.Serializable]  
public class Boundary  
{  
 public float xMin, xMax, zMin, zMax;

private Rigidbody rb;

Void Start(){

rb= GetComponent<Rigidbody>();

}  
}  
  
public class PlayerController : MonoBehaviour  
{  
 public float speed;  
 public float tilt;  
 public Boundary boundary;  
  
 void FixedUpdate ()  
 {  
 float moveHorizontal = [Input.GetAxis](http://docs.unity3d.com/Documentation/ScriptReference/Input.GetAxis.html) ("Horizontal");  
 float moveVertical = [Input.GetAxis](http://docs.unity3d.com/Documentation/ScriptReference/Input.GetAxis.html) ("Vertical");  
  
 Vector3 movement = new Vector3 (moveHorizontal, 0.0f, moveVertical);  
 rb.velocity = movement \* speed;  
  
 rb.position = new Vector3   
 (  
  [Mathf.Clamp](http://docs.unity3d.com/Documentation/ScriptReference/Mathf.Clamp.html) (rigidbody.position.x, boundary.xMin, boundary.xMax),   
 0.0f,   
  [Mathf.Clamp](http://docs.unity3d.com/Documentation/ScriptReference/Mathf.Clamp.html) (rigidbody.position.z, boundary.zMin, boundary.zMax)  
 );  
  
 rb.rotation = Quaternion.Euler (0.0f, 0.0f, rigidbody.velocity.x \* -tilt);  
 }  
}

I continued to finish the tutorial for creating shots and shooting shots. The code for the mover for creating shots was :

using UnityEngine;  
using System.Collections;  
  
public class Mover : MonoBehaviour  
{  
 public float speed;  
  
 void Start ()  
 {  
 rigidbody.velocity = transform.forward \* speed;  
 }  
}

I finished working on the rest of the Space Shooter tutorials which consisted of creating a boundary for the game, the player, and the objects of the game, creating hazards, creating explosions for when the space shooter shoots an object, coding the program for the game controller and modifying how the player can change/play the game, and also making spawning waves. I also did the tutorial for the audio, counting and displaying the points on to the screen, and ending and building the game.

Also, as Professor Seidenstein suggested to research, I looked for patents for Chemtris on Google Patents and didn’t find any. There was an extensive amount of resources that can assist with academic patenting or patenting the research/app/name through the university and they can be found here:

<http://www.wipo.int/sme/en/documents/academic_patenting.html>

<http://www.tms.org/pubs/journals/JOM/matters/matters-0301.html>

**Week 2: Week of June 12th**

*Day 1: June 12th:*

I continued to begin working on the other tutorials available on the Unity website. There were two tutorials based solely on 2D games, which were the 2D Roguelike tutorial and the 2D UFO tutorial. Since our project is focused on 3D and not 2D, i felt that doing these tutorials would not be helpful and would even make learning Unity a bit more confusing so I didn't do them. I worked on the Survival shooter tutorial and worked on how to set up the environment, design the player character, setup the camera, create the enemy #1, health HUD, player health, harming the enemy, scoring points, spawning enemies and determining at what point the game is over and how to show on the screen that the game is over.

For the player movement, the relevant code was:

using UnityEngine;  
  
public class PlayerMovement : MonoBehaviour  
{  
 public float speed = 6f; // The speed that the player will move at.  
  
 Vector3 movement; // The vector to store the direction of the player's movement.  
 Animator anim; // Reference to the animator component.  
 Rigidbody playerRigidbody; // Reference to the player's rigidbody.  
 int floorMask; // A layer mask so that a ray can be cast just at gameobjects on the floor layer.  
 float camRayLength = 100f; // The length of the ray from the camera into the scene.  
  
 void Awake ()  
 {  
 // Create a layer mask for the floor layer.  
 floorMask = LayerMask.GetMask ("Floor");  
  
 // Set up references.  
 anim = GetComponent <Animator> ();  
 playerRigidbody = GetComponent <Rigidbody> ();  
 }  
  
  
 void FixedUpdate ()  
 {  
 // Store the input axes.  
 float h = [Input.GetAxisRaw](http://docs.unity3d.com/Documentation/ScriptReference/Input.GetAxisRaw.html) ("Horizontal");  
 float v = [Input.GetAxisRaw](http://docs.unity3d.com/Documentation/ScriptReference/Input.GetAxisRaw.html) ("Vertical");  
  
 // Move the player around the scene.  
 Move (h, v);  
  
 // Turn the player to face the mouse cursor.  
 Turning ();  
  
 // Animate the player.  
 Animating (h, v);  
 }  
  
 void Move (float h, float v)  
 {  
 // Set the movement vector based on the axis input.  
 movement.Set (h, 0f, v);  
   
 // Normalise the movement vector and make it proportional to the speed per second.  
 movement = movement.normalized \* speed \* [Time.deltaTime](http://docs.unity3d.com/Documentation/ScriptReference/Time-deltaTime.html);  
  
 // Move the player to it's current position plus the movement.  
 playerRigidbody.MovePosition (transform.position + movement);  
 }  
  
 void Turning ()  
 {  
 // Create a ray from the mouse cursor on screen in the direction of the camera.  
 Ray camRay = Camera.main.ScreenPointToRay ([Input.mousePosition](http://docs.unity3d.com/Documentation/ScriptReference/Input-mousePosition.html));  
  
 // Create a RaycastHit variable to store information about what was hit by the ray.  
 RaycastHit floorHit;  
  
 // Perform the raycast and if it hits something on the floor layer...  
 if(Physics.Raycast (camRay, out floorHit, camRayLength, floorMask))  
 {  
 // Create a vector from the player to the point on the floor the raycast from the mouse hit.  
 Vector3 playerToMouse = floorHit.point - transform.position;  
  
 // Ensure the vector is entirely along the floor plane.  
 playerToMouse.y = 0f;  
  
 // Create a quaternion (rotation) based on looking down the vector from the player to the mouse.  
 Quaternion newRotation = Quaternion.LookRotation (playerToMouse);  
  
 // Set the player's rotation to this new rotation.  
 playerRigidbody.MoveRotation (newRotation);  
 }  
 }  
  
 void Animating (float h, float v)  
 {  
 // Create a boolean that is true if either of the input axes is non-zero.  
 bool walking = h != 0f || v != 0f;  
  
 // Tell the animator whether or not the player is walking.  
 anim.SetBool ("IsWalking", walking);  
 }  
}

And the code for the player health was as follows:

using UnityEngine;  
using UnityEngine.UI;  
using System.Collections;  
  
public class PlayerHealth : MonoBehaviour  
{  
 public int startingHealth = 100; // The amount of health the player starts the game with.  
 public int currentHealth; // The current health the player has.  
 public Slider healthSlider; // Reference to the UI's health bar.  
 public Image damageImage; // Reference to an image to flash on the screen on being hurt.  
 public AudioClip deathClip; // The audio clip to play when the player dies.  
 public float flashSpeed = 5f; // The speed the damageImage will fade at.  
 public Color flashColour = new Color(1f, 0f, 0f, 0.1f); // The colour the damageImage is set to, to flash.  
  
  
 Animator anim; // Reference to the Animator component.  
 AudioSource playerAudio; // Reference to the AudioSource component.  
 PlayerMovement playerMovement; // Reference to the player's movement.  
 PlayerShooting playerShooting; // Reference to the PlayerShooting script.  
 bool isDead; // Whether the player is dead.  
 bool damaged; // True when the player gets damaged.  
  
  
 void Awake ()  
 {  
 // Setting up the references.  
 anim = GetComponent <Animator> ();  
 playerAudio = GetComponent <AudioSource> ();  
 playerMovement = GetComponent <PlayerMovement> ();  
 playerShooting = GetComponentInChildren <PlayerShooting> ();  
  
 // Set the initial health of the player.  
 currentHealth = startingHealth;  
 }  
  
  
 void Update ()  
 {  
 // If the player has just been damaged...  
 if(damaged)  
 {  
 // ... set the colour of the damageImage to the flash colour.  
 damageImage.color = flashColour;  
 }  
 // Otherwise...  
 else  
 {  
 // ... transition the colour back to clear.  
 damageImage.color = [Color.Lerp](http://docs.unity3d.com/Documentation/ScriptReference/Color.Lerp.html) (damageImage.color, [Color.clear](http://docs.unity3d.com/Documentation/ScriptReference/Color-clear.html), flashSpeed \* [Time.deltaTime](http://docs.unity3d.com/Documentation/ScriptReference/Time-deltaTime.html));  
 }  
  
 // Reset the damaged flag.  
 damaged = false;  
 }  
  
  
 public void TakeDamage (int amount)  
 {  
 // Set the damaged flag so the screen will flash.  
 damaged = true;  
  
 // Reduce the current health by the damage amount.  
 currentHealth -= amount;  
  
 // Set the health bar's value to the current health.  
 healthSlider.value = currentHealth;  
  
 // Play the hurt sound effect.  
 playerAudio.Play ();  
  
 // If the player has lost all it's health and the death flag hasn't been set yet...  
 if(currentHealth <= 0 && !isDead)  
 {  
 // ... it should die.  
 Death ();  
 }  
 }  
  
  
 void Death ()  
 {  
 // Set the death flag so this function won't be called again.  
 isDead = true;  
  
 // Turn off any remaining shooting effects.  
 playerShooting.DisableEffects ();  
  
 // Tell the animator that the player is dead.  
 anim.SetTrigger ("Die");  
  
 // Set the audiosource to play the death clip and play it (this will stop the hurt sound from playing).  
 playerAudio.clip = deathClip;  
 playerAudio.Play ();  
  
 // Turn off the movement and shooting scripts.  
 playerMovement.enabled = false;  
 playerShooting.enabled = false;  
 }   
}

I also did the Tanks tutorial which is a 2 player 1 keyboard shooter game. The tutorial taught how to setup the scene, create the tank and write code to control the tank, control the camera, write code to control the tank’s health, create shells, fire shells, manage games, and mix the audio for the background of the game. I also tried to do the adventure game tutorial which is a 6 phase adventure game. The tutorial teaches you how to create the player, work on the inventory, set up the conditions, and code the reactions, interactables and game state. I was able to work on both the Tanks tutorial and the Adventure game tutorials because the tutorials were relatively short (7 videos each) compared to the Space shooter tutorial which was 19 videos.

*Day 2: June 13th:*

I kept in touch with Mohamed, the work-study student who is currently working on the project and I discussed a couple of ideas with him to make the game more educational. Our design ideas as of now consist of introducing topics of hydrophobicity and hydrophilicity and relating it back to diseases that stem from problems in hydrophobicity/hydrophilicity (sickle cell anemia). We could do this by having different compounds available and preventing the player from connecting a non polar molecule with water to show hydrophobicity or only allowing two polar molecules/two non polar molecules to be connected with each other to show the concept of "like dissolves like." For example to make a "tetromino" piece non-polar, we would have to create a variable called Polarity and set it to false and we can then control which molecules this piece is allowed to interact with. In order to prevent ourselves from making Tetris, which is already a difficult game, hard to play by adding too many concepts, we have to figure out how to implement these ideas in a way that makes the game simple and fluid. Instead of making the user lose the game if they try to attach a polar molecule to a non-polar molecule, we were thinking we could implement a penalty of some sort that would result in the user losing points. This would make the game more educational than frustrating.

We were also thinking of introducing topics of chirality by intertwining it with the biological importance of chirality (what happens if you ingest the wrong chiral form of a vitamin etc.) to make it more educational (we could implement it as a bonus level maybe). According to Mohamed, the coding for the base of the game is complete but we need to use game mechanics like polarity, chirality, and hydrophobicity.

We were also thinking that another long-term goal would be to make the game adaptable to iOS or possibly make a VR version of it. Making an iOS version would be relatively simple because Unity has the option of switching the build type between Android and iOS. The VR version is a little more complex and would require a lot more work and collaboration but I think it would be feasible.

We discussed all of these ideas with Prof. Tommy Lee and took down his feedback. His feedback was that he wanted Mohamed to work on converting all the libGDX code to unity and for him to convert the game from a spherical form to a cubical 3D layout and he would like Eugene and I to work on making the different levels. Level 1 would be a 1 by 1 by 1 cube on a 3 by 3 grid and level 2 would be a 1 by 1 by 1 cube on a 1 by 2 by 1 grid etc. The last level would be a 6 by 6 by 6 grid. He wants us to work on this for the next two weeks and after that, he wants us to go into the game mechanics such as the polarity, chirality, and hydrophobicity of the game.

The difficult part about working on this project is that there are so many contributors to this project. There’s Eugene and I who are working on it for the summer, there’s Mohamed who is the work-study student working on it during the semester, and Adrian who is the main contributor to the project and we currently do not have access to all the code. Since a main part of the project is converting the game from libGDX to Unity, it is a little difficult to do so without having all the relevant code.

*Day 3: June 14th:*

I started reading through different journal articles to see if anything would help with the coding and design aspects of the game. I read multiple articles by Michele D. Dickey, which I found through NYU’s access to JSTOR. Michele D. Dickey seems to be an expert in the field that we’re trying to make Chemtris prominent in, which is the intersection of education and game design. I found her article “Engaging by Design: How Engagement Strategies In Popular Computer and Video Games Can Inform Instructional Design” to be extremely helpful in clarifying our goals for this research program. Throughout her article she shows how useful effective game design in video and computer games can be in promoting engaged learning in a classroom setting. I loved how she was unbiased throughout her article and properly provided the reader with all the information they needed to come to a conclusion of their own. She thoroughly investigates the effects of different design elements from popular computer and video games on the creation of a fun and engaging learning environment. She notes how important an interactive design format is to enable an ideal learning environment and discusses the benefits of a good setting, roles and characters, actions and feedback, perspective, backstory, and challenging tasks throughout a game. According to her, having occasional affirmations of performance, role-playing and clear and compelling standards is key to a fun but educational game. This was extremely helpful because Chemtris’ ultimate goal is to make children WANT to learn more about organic chemistry, inorganic chemistry and biology. Our goal is to make this game as fun and educational as possible and this journal article really helped me get a better idea of what we need to do and how we need to do it.

I worked on a couple of the assignments on the Unity website to get a better feel for the software. The assignments were very comprehensive and so they also helped me with understanding the game mechanics and design aspects a little better. I worked on the “Making an ‘Angry Birds’ style game which goes over 2D Physics in detail such as Rigidbody, Collider, Hinge Joint, Area Effector, Distance Joint, Point Effector, and Bouncing & Sliding in 2D.

*Day 4: June 15th:*

The last time I spoke with Prof. Lee, it was a little helpful but the details of the goals of the game and how he wants us to go about structuring the levels was still a little confusing so I went and talked with him again today. Level 1 is a 1 by 1 by 1 cube on a 3 by 3 grid. Level 2 introduces two new pieces on a 5 by 5 by 5 grid, one of the two new pieces is the original 1 by 1 by 1 cube and the other piece is the rectangle which is 1 by 2 by 1. Level 3 is the same 5 by 5 by 5 grid but three pieces, two of which are the same ones from level 2 and the third piece is the rectangle with the cube attached to it. Level 4 involves the plus sign pieces. Level 5 involves the donut piece. Levels 6-10 is on a bigger grid (6 by 6 by 6) or the grid can start off at 6 by 6 by 6 and get progressively bigger. Level 9 will include the z shaped pieces and introduce chirality. Level 10 will include the red and blue colors of the different z shaped pieces and will introduce polarity. and hydrophobicity/hydrophilicity will be introduced later on in the more advanced levels like Level 15. So the main goals right now are to add colors (make the colors pop like in candy crush) and make the rotation easier, because right now it's 360 degree rotation which makes it harder and he just wants it to be 90 or 120 degree turns on both sides and a top and bottom view. Before the first WIP, our main goals are to work on converting all the code into Unity and changing the format from spherical to cubical as well as implementing levels 1 and 2.

I read through more of the research articles written by Michele D. Dickey. I read through her article “Murder on Grimm Isle: The impact of game narrative design in an educational game-based learning environment” because I thought it would be useful in determining the factors that have a significant impact on the educational effects of video games. However, by the end of the article, I realized the article mainly focuses on the effects of narratives on video games and the specific video game we’re working on doesn't really require a strong narrative. I continued to read through more of her articles and found her article “Game Design Narrative for Learning: Appropriating Adventure Game Design Narrative Devices and Techniques for the Design of Interactive Learning Environments” to be extremely interesting and useful. She goes into detail in her conceptual analytical research of educational games by examining the role of narrative in adventure game design and also analyzes the optimal structure and design framework that results in the most effective interactive learning environments for children.

*Day 5: June 16th:*

The first hour and a half of the day was the Graduate Admissions event with guest speaker Mallika Grayson who spoke about her experiences in research and further education (graduate school and PhD). I found this event extremely interactive, especially where she made us go into groups and discuss possible obstacles and come up with possible solutions to those obstacles. This allowed me to meet 5 to 6 other students in the program who I haven’t met before and we collaborated to come up with ideas for both exercises. I also found the tips that she gave about finding research labs, networking effectively, applying to grad school and PhD programs, and career paths as well as the general advice she gave us about life and learning to be very useful and worthwhile.

After the event, I spent the majority of the day working on researching more articles and reading chapters from books about game design and the Unity platform. The book Unity Game Development Essentials by Will Goldstone was helpful in providing a clear and concise overview of each aspect of Unity. Link here: <https://books.google.com/books?hl=en&lr=&id=WfAWzVW9IK0C&oi=fnd&pg=PT16&dq=unity+game+engine&ots=ATbTHDIEDf&sig=9HXXWc6iPakIY2UtVLYImAG8fL8#v=onepage&q=unity%20game%20engine&f=false>

Specific chapters from the book Beginning 3D Game Development with Unity 4: All-in-one, Multi-Platform game Development such as Chapter 5 on Navigation and Functionality and Chapter 11 on Physics and Special Effects were helpful because I will need to know game mechanics and how to use physics in the code to implement the movement and speed of the game objects as well as collisions between objects.

**Week 3: Week of June 19th**

*Day 1: June 19th:*

Mohamed shared the GitHub repository with all the relevant code, both for the circular tetris and the base for the cubical format that is required for this version of the game. I started modifying the code for the game menu which will be the first thing the player sees as soon as they start the game.

The code for the game menu is the following:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement; *//in order to load scenemanager*

using UnityEngine.UI; *//to update text*

public class GameMenu : MonoBehaviour {

public Text levelText;

*// Use this for initialization*

void Start () {

levelText.text = "0";

}

public void PlayGame () {

if (Game.startingLevel == 0) {

Game.startingAtLevelZero = true;

} else {

Game.startingAtLevelZero = false;

}

SceneManager.LoadScene ("level1");

}

public void ChangedLevel (float value) {

Game.startingLevel = (int)value;

levelText.text = value.ToString ();

}

public void LaunchGameMenu(){

SceneManager.LoadScene ("gamemenu");

}

}

The code currently has Level 0 (1x1x1 cube on a 3x3 grid)

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement; *//in order to load scenemanager*

using UnityEngine.UI; *//in order to control score text object*

public class Game : MonoBehaviour {

public static int gridWidth = 3; *//max width of grid (Z as well)*

public static int gridWidthtemp; *//used to re-update grid values once it is expanded*

public static int gridHeight = 15; *//max height of grid*

public static Transform[ , , ] grid; *//initialize grid variable (It is a 3D-array)*

public static Transform[ , , ] gridtemp; *//used to re-size grid once it is expanded*

private GameObject stage; *//used to indicate min position of a tetromino (the stage's position).*

private Camera mainCam; *//used to move camera after grid is expanded, so as to assure it is centered.*

private bool expandedGrid = false; *//not used (but can be used to create a more general grid expansion function).*

*//score variables*

public int scoreOneLine = 40;

public int scoreTwoLine = 100;

public int scoreThreeLine = 300;

public int scoreFourLine = 1200;

private int numberOfRowsThisTurn = 0;

public Text hud\_score;

public static int currentScore = 0;

*//score variables end*

*//gui text for level and lines cleared*

public Text hud\_level;

public Text hud\_lines;

*//level variables*

public float fallSpeed = 1.0f; *//speed of tetromino (decreases with level increase)*

public static bool startingAtLevelZero; *//used to implement level slider*

public static int startingLevel; *//used to implement level slider*

public int currentLevel = 0;

private int numOfRowsCleared = 0;

*//level variables end*

*//public GameObject[] Tetrominos; This is the OLD array implementation of tetrominos*

List<GameObject> Tetrominos = new List<GameObject>(); *//list of active tetrominos falling*

GameObject startCube; *//starting tetromino that is added to list in start function*

private GameObject ghostTetromino; *//preview/ghost tetromino (shows where piece is falling)*

private GameObject nextTetromino; *//next tetromino that is falling*

private GameObject previewTetromino; *//identifies next tetromino and displays it in the background*

public GameObject liveTetromino; *//this is used for the moveButtons*

private Vector3 previewTetrominoPosition = new Vector3 (-2f, 1, 5f); *//location of preview tetromino*

private bool gameStarted = false; *//Variable to identify whether the game just started, or not. (To ensure that later in the game*

*//the preview tetromino becomes the next tetromino.) i.e. in the start we have no tetromino preview to have to land, but later on we do.*

void Start () {

currentLevel = startingLevel;

*//At start of game, add 1x1x1 tetromino prefab to tetrominos list*

startCube = Resources.Load("Prefabs/SingleCube") as GameObject;

Tetrominos.Add (startCube);

SpawnNextTetromino ();

*//I was playing with scenes, so if starting scene is "level1" then must have width of 3 as grid is 3x3*

*//instead of expanding grid on the spot, we can simply switch to a different scene when levelling up (but this*

*//creates a slight delay).*

Scene scene = SceneManager.GetActiveScene();

if (scene.name == "level1") {

gridWidth = 3;

}

*//instantiating grid array to keep track of which positions are available for a tetromino to land on*

*//this can be there in the initialization list at the beginning, but in case we need to alter gridWidth in start function*

*//I decided to place it here*

grid = new Transform[gridWidth, gridHeight, gridWidth];

}

*Day 2: June 20th:*

Continued to work on the code for Level 1 (1x1x1 cube and 1x2x1 cube on a 4x4 grid).

*//function used to expandgrid based on level*

void ExpandGrid()

{

if (currentLevel == 1 && GameObject.Find("upgradeto4x4(Clone)") != true) { *//if level is 1 and grid is not already upgraded to a 4x4 grid*

Vector3 spawnPos = new Vector3(3f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

Quaternion spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

GameObject gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto4x4", typeof(GameObject)), spawnPos, spawnRot);

*//make grid upgrade prefab child of base object*

stage = GameObject.FindWithTag("Base");

gridexpansion.transform.parent = stage.transform;

*//Copy old gridwidth into temp val*

gridWidthtemp = gridWidth;

*//Update grid width to 4*

gridWidth = 4;

*//create new grid array*

gridtemp = new Transform[gridWidth, gridHeight, gridWidth];

*//switch over positions (to keep track of positions that are already taken up).*

for (int y = 0; y < gridHeight; y++) {

for (int x = 0; x < gridWidthtemp; x++) {

for (int z = 0; z < gridWidthtemp; z++) {

gridtemp [x, y, z] = grid [x, y, z];

}

}

}

*//set grid variable equal to new resized grid*

grid = gridtemp;

*//add 1x2x1 tetromino prefab to tetrominos list so it begins falling*

Tetrominos.Add (Resources.Load ("Prefabs/DoubleCube") as GameObject);

*//Add 0.5 to the cameras X position*

mainCam = Camera.main; *// Grab a reference to the camera*

mainCam.transform.position += new Vector3(0.5f, 0, 0);

}

I also spent some time reading through the comments for the article reviews from last week and I researched some more relevant articles for my project and worked on the article reviews. I found an article about using three-dimensional molecular models to teach molecular structures in high school chemistry and found it to be extremely useful because it shows exactly why we need Chemtris and why Chemtris fills a void in science education. Science education is almost completely dominated by two-dimensional molecular models in textbooks or by three-dimensional mechanical molecular models. However, both are fairly ineffective because they don't completely show all the dimensional properties of molecules which is why computer generated three-dimensional molecular models are needed to do justice to all the chemical and physical properties of molecules especially intermolecular and intramolecular interactions. Link here: <https://link.springer.com/content/pdf/10.1007%2FBF02211261.pdf>

*Day 3: June 21st:*

Continued to work on the code for Level 2 and Level 3. The code for Levels 2 and 3 consisted of basically else if statements to check the current level and involved using a stacked for loop to change the grid size to the respective size (level 2 included a 5x5 grid size and level 3 had the same grid size but introduced a new tetromino which was the L shaped tetromino (1x2x1 tetromino with a cube attached to it):

else if (currentLevel == 2 && GameObject.Find("upgradeto5x5(Clone)") != true) {

Vector3 spawnPos = new Vector3(4f, 0f, 0f);

Quaternion spawnRot = Quaternion.identity;

GameObject gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto5x5", typeof(GameObject)), spawnPos, spawnRot);

stage = GameObject.FindWithTag("Base");

gridexpansion.transform.parent = stage.transform;

gridWidthtemp = gridWidth;

gridWidth = 5;

gridtemp = new Transform[gridWidth, gridHeight, gridWidth];

for (int y = 0; y < gridHeight; y++) {

for (int x = 0; x < gridWidthtemp; x++) {

for (int z = 0; z < gridWidthtemp; z++) {

gridtemp [x, y, z] = grid [x, y, z];

}

}

}

grid = gridtemp;

mainCam = Camera.main; *// Grab a reference to the camera\*

mainCam.transform.position += new Vector3(0.5f, 0, 0);

*//The following code is in case a user uses the menu slider to go to a level manually, it ensures*

*//that the stage is expanded completely to this level.*

if (GameObject.Find ("upgradeto4x4(Clone)") != true) {

spawnPos = new Vector3(3f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto4x4", typeof(GameObject)), spawnPos, spawnRot);

}

}

else if (currentLevel == 3) {

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SmallLCube") as GameObject)) { *//if tetromino list does not contain*

*//the tetromino piece we want to add, then add it.*

Tetrominos.Add (Resources.Load ("Prefabs/SmallLCube") as GameObject);

}

Vector3 spawnPos = new Vector3(4f, 0f, 0f);

Quaternion spawnRot = Quaternion.identity;

GameObject gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto5x5", typeof(GameObject)), spawnPos, spawnRot);

stage = GameObject.FindWithTag("Base");

gridexpansion.transform.parent = stage.transform;

gridWidthtemp = gridWidth;

gridWidth = 5;

gridtemp = new Transform[gridWidth, gridHeight, gridWidth];

for (int y = 0; y < gridHeight; y++) {

for (int x = 0; x < gridWidthtemp; x++) {

for (int z = 0; z < gridWidthtemp; z++) {

gridtemp [x, y, z] = grid [x, y, z];

}

}

}

grid = gridtemp;

*//The following code is in case a user uses the menu slider to go to a level manually, it ensures*

*//that the stage is expanded completely to this level.*

if (GameObject.Find ("upgradeto4x4(Clone)") != true) {

spawnPos = new Vector3(3f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto4x4", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto5x5(Clone)") != true) {

spawnPos = new Vector3(4f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto5x5", typeof(GameObject)), spawnPos, spawnRot);

}

*//The following code ensures the tetrominos list has all the tetrominos needed.*

*//If the user decides to pick a level manually.*

if (!Tetrominos.Contains (Resources.Load ("Prefabs/DoubleCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/DoubleCube") as GameObject);

}

}

Today, we also had the career development panel with 4 female engineers in the workforce and I thought it was one of the most valuable, interesting panels I’ve ever been to. Everyone there was so interesting, knowledgeable and passionate about their fields and they gave some really great advice about picking a career choice, not being so caught up on the idea that what job you get after college determines your entire life and career trajectory because it doesn't and also about how to succeed as a woman in STEM and how to play out your strengths and not your weaknesses. I also got to talk a bit with Patricia, the biomedical engineer panelist, afterwards and she was very encouraging and sweet and also gave some great advice. It was a wonderful experience overall, I really loved it a lot.

*Day 4: June 22nd:*

Continued to work on the code for levels 4, 5, and 6 of the project:

else if (currentLevel == 4) {

if (!Tetrominos.Contains (Resources.Load ("Prefabs/TCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/Tcube") as GameObject);

}

Vector3 spawnPos = new Vector3(4f, 0f, 0f);

Quaternion spawnRot = Quaternion.identity;

GameObject gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto5x5", typeof(GameObject)), spawnPos, spawnRot);

stage = GameObject.FindWithTag("Base");

gridexpansion.transform.parent = stage.transform;

gridWidthtemp = gridWidth;

gridWidth = 5;

gridtemp = new Transform[gridWidth, gridHeight, gridWidth];

for (int y = 0; y < gridHeight; y++) {

for (int x = 0; x < gridWidthtemp; x++) {

for (int z = 0; z < gridWidthtemp; z++) {

gridtemp [x, y, z] = grid [x, y, z];

}

}

}

grid = gridtemp;

*//The following code is in case a user uses the menu slider to go to a level manually, it ensures*

*//that the stage is expanded completely to this level.*

if (GameObject.Find ("upgradeto4x4(Clone)") != true) {

spawnPos = new Vector3(3f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto4x4", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto5x5(Clone)") != true) {

spawnPos = new Vector3(4f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto5x5", typeof(GameObject)), spawnPos, spawnRot);

}

*//The following code ensures the tetrominos list has all the tetrominos needed.*

*//If the user decides to pick a level manually.*

if (!Tetrominos.Contains (Resources.Load ("Prefabs/DoubleCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/DoubleCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SmallLCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SmallLCube") as GameObject);

}

}

else if (currentLevel == 5) {

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SquareCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SquareCube") as GameObject);

}

Vector3 spawnPos = new Vector3(4f, 0f, 0f);

Quaternion spawnRot = Quaternion.identity;

GameObject gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto5x5", typeof(GameObject)), spawnPos, spawnRot);

stage = GameObject.FindWithTag("Base");

gridexpansion.transform.parent = stage.transform;

gridWidthtemp = gridWidth;

gridWidth = 5;

gridtemp = new Transform[gridWidth, gridHeight, gridWidth];

for (int y = 0; y < gridHeight; y++) {

for (int x = 0; x < gridWidthtemp; x++) {

for (int z = 0; z < gridWidthtemp; z++) {

gridtemp [x, y, z] = grid [x, y, z];

}

}

}

grid = gridtemp;

*//The following code is in case a user uses the menu slider to go to a level manually, it ensures*

*//that the stage is expanded completely to this level.*

if (GameObject.Find ("upgradeto4x4(Clone)") != true) {

spawnPos = new Vector3(3f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto4x4", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto5x5(Clone)") != true) {

spawnPos = new Vector3(4f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto5x5", typeof(GameObject)), spawnPos, spawnRot);

}

*//The following code ensures the tetrominos list has all the tetrominos needed.*

*//If the user decides to pick a level manually.*

if (!Tetrominos.Contains (Resources.Load ("Prefabs/DoubleCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/DoubleCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SmallLCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SmallLCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/TCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/TCube") as GameObject);

}

}

else if (currentLevel == 6 && GameObject.Find("upgradeto6x6(Clone)") != true) {

Vector3 spawnPos = new Vector3(0f, 0f, 5f);

Quaternion spawnRot = Quaternion.identity;

GameObject gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto6x6", typeof(GameObject)), spawnPos, spawnRot);

stage = GameObject.FindWithTag("Base");

gridexpansion.transform.parent = stage.transform;

gridWidthtemp = gridWidth;

gridWidth = 6;

gridtemp = new Transform[gridWidth, gridHeight, gridWidth];

for (int y = 0; y < gridHeight; y++) {

for (int x = 0; x < gridWidthtemp; x++) {

for (int z = 0; z < gridWidthtemp; z++) {

gridtemp [x, y, z] = grid [x, y, z];

}

}

}

grid = gridtemp;

mainCam = Camera.main; *// Grab a reference to the camera\*

mainCam.transform.position += new Vector3(0.5f, 0, 0);

*//The following code is in case a user uses the menu slider to go to a level manually, it ensures*

*//that the stage is expanded completely to this level.*

*//Also, it ensures the tetrominos list has all the tetrominos needed.*

if (GameObject.Find ("upgradeto4x4(Clone)") != true) {

spawnPos = new Vector3(3f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto4x4", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto5x5(Clone)") != true) {

spawnPos = new Vector3(4f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto5x5", typeof(GameObject)), spawnPos, spawnRot);

}

*//The following code ensures the tetrominos list has all the tetrominos needed.*

*//If the user decides to pick a level manually.*

if (!Tetrominos.Contains (Resources.Load ("Prefabs/DoubleCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/DoubleCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SmallLCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SmallLCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/TCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/TCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SquareCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SquareCube") as GameObject);

}

}

Also continued to read through more relevant articles for the project and think of design elements that we can implement. Ideally, we’re trying to work together as a group (Mohamed, Eugene and I) to finish the base of the game, with the optimal colors as well, before the next week so we can have a couple people play the game and incorporate their feedback into the game to make it more fun and more educational.

*Day 5: June 23rd:*

I researched more articles that would be relevant to my project. I found an article called Promoting Understanding of Chemical Representations: Students' Use of a Visualization Tool in the Classroom which talks about a computer program called eChem that is similar to Chemtris in that it also is a computational three-dimensional molecular modeling tool that is useful in teaching concepts of organic and inorganic chemistry but different in that eChem is strictly an educational classroom tool while Chemtris is a video game that makes learning chemistry fun and interactive but can also be implemented in a classroom setting with supplemental tools to promote learning. Throughout this article, Hsin-Kai Wu, Joseph S. Krajcik, and Elliot Soloway analyze the results of a study that was used to test the retention of science material in children before and after using eChem. The results showed that eChem was highly effective in a educational setting and caused students to learn more and retain more information because eChem encourages students to construct models while making referential linkages between the visual representations and the conceptual representations they were exposed to. It also allows students to translate representations back and forth between two and three dimensional molecular models. I found this article to be useful in developing Chemtris because it gave us a framework for what we’re trying to achieve with Chemtris but also encouraged us to think of more ways we can make Chemtris better and more different than existing computational three-dimensional molecular modeling platforms that are already out there.

Link to article: <http://onlinelibrary.wiley.com/doi/10.1002/tea.1033/epdf>

Finished the code for the rest of the game, levels 7 to 10:

else if (currentLevel == 7 && GameObject.Find("upgradeto7x7(Clone)") != true) {

else if (currentLevel == 7 && GameObject.Find("upgradeto7x7(Clone)") != true) {

Vector3 spawnPos = new Vector3(0f, 0f, 6f);

Quaternion spawnRot = Quaternion.identity;

GameObject gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto7x7", typeof(GameObject)), spawnPos, spawnRot);

stage = GameObject.FindWithTag("Base");

gridexpansion.transform.parent = stage.transform;

gridWidthtemp = gridWidth;

gridWidth = 7;

gridtemp = new Transform[gridWidth, gridHeight, gridWidth];

for (int y = 0; y < gridHeight; y++) {

for (int x = 0; x < gridWidthtemp; x++) {

for (int z = 0; z < gridWidthtemp; z++) {

gridtemp [x, y, z] = grid [x, y, z];

}

}

}

grid = gridtemp;

mainCam = Camera.main; *// Grab a reference to the camera\*

mainCam.transform.position += new Vector3(0.5f, 0, 0);

*//The following code is in case a user uses the menu slider to go to a level manually, it ensures*

*//that the stage is expanded completely to this level.*

*//Also, it ensures the tetrominos list has all the tetrominos needed.*

if (GameObject.Find ("upgradeto4x4(Clone)") != true) {

spawnPos = new Vector3(3f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto4x4", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto5x5(Clone)") != true) {

spawnPos = new Vector3(4f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto5x5", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto6x6(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 5f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto6x6", typeof(GameObject)), spawnPos, spawnRot);

}

*//check prev levels*

if (!Tetrominos.Contains (Resources.Load ("Prefabs/DoubleCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/DoubleCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SmallLCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SmallLCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/TCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/TCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SquareCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SquareCube") as GameObject);

}

}

else if (currentLevel == 8 && GameObject.Find("upgradeto8x8(Clone)") != true) {

Vector3 spawnPos = new Vector3(0f, 0f, 7f);

Quaternion spawnRot = Quaternion.identity;

GameObject gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto8x8", typeof(GameObject)), spawnPos, spawnRot);

stage = GameObject.FindWithTag("Base");

gridexpansion.transform.parent = stage.transform;

gridWidthtemp = gridWidth;

gridWidth = 8;

gridtemp = new Transform[gridWidth, gridHeight, gridWidth];

for (int y = 0; y < gridHeight; y++) {

for (int x = 0; x < gridWidthtemp; x++) {

for (int z = 0; z < gridWidthtemp; z++) {

gridtemp [x, y, z] = grid [x, y, z];

}

}

}

grid = gridtemp;

mainCam = Camera.main; *// Grab a reference to the camera\*

mainCam.transform.position += new Vector3(0.5f, 0, 0);

*//The following code is in case a user uses the menu slider to go to a level manually, it ensures*

*//that the stage is expanded completely to this level.*

*//Also, it ensures the tetrominos list has all the tetrominos needed.*

if (GameObject.Find ("upgradeto4x4(Clone)") != true) {

spawnPos = new Vector3(3f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto4x4", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto5x5(Clone)") != true) {

spawnPos = new Vector3(4f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto5x5", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto6x6(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 5f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto6x6", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto7x7(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 6f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto7x7", typeof(GameObject)), spawnPos, spawnRot);

}

*//The following code ensures the tetrominos list has all the tetrominos needed.*

*//If the user decides to pick a level manually.*

if (!Tetrominos.Contains (Resources.Load ("Prefabs/DoubleCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/DoubleCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SmallLCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SmallLCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/TCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/TCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SquareCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SquareCube") as GameObject);

}

}

else if (currentLevel == 9 && GameObject.Find("upgradeto9x9(Clone)") != true) {

Vector3 spawnPos = new Vector3(0f, 0f, 8f);

Quaternion spawnRot = Quaternion.identity;

GameObject gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto9x9", typeof(GameObject)), spawnPos, spawnRot);

stage = GameObject.FindWithTag("Base");

gridexpansion.transform.parent = stage.transform;

gridWidthtemp = gridWidth;

gridWidth = 9;

gridtemp = new Transform[gridWidth, gridHeight, gridWidth];

for (int y = 0; y < gridHeight; y++) {

for (int x = 0; x < gridWidthtemp; x++) {

for (int z = 0; z < gridWidthtemp; z++) {

gridtemp [x, y, z] = grid [x, y, z];

}

}

}

grid = gridtemp;

mainCam = Camera.main; *// Grab a reference to the camera\*

mainCam.transform.position += new Vector3(0.5f, 0, 0);

if (!Tetrominos.Contains (Resources.Load ("Prefabs/ZCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/ZCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SCube") as GameObject);

}

*//The following code is in case a user uses the menu slider to go to a level manually, it ensures*

*//that the stage is expanded completely to this level.*

*//Also, it ensures the tetrominos list has all the tetrominos needed.*

if (GameObject.Find ("upgradeto4x4(Clone)") != true) {

spawnPos = new Vector3(3f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto4x4", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto5x5(Clone)") != true) {

spawnPos = new Vector3(4f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto5x5", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto6x6(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 5f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto6x6", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto7x7(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 6f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto7x7", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto8x8(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 7f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto8x8", typeof(GameObject)), spawnPos, spawnRot);

}

*//The following code ensures the tetrominos list has all the tetrominos needed.*

*//If the user decides to pick a level manually.*

if (!Tetrominos.Contains (Resources.Load ("Prefabs/DoubleCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/DoubleCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SmallLCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SmallLCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/TCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/TCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SquareCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SquareCube") as GameObject);

}

}

else if (currentLevel == 10 && GameObject.Find("upgradeto10x10(Clone)") != true) {

Vector3 spawnPos = new Vector3(0f, 0f, 9f);

Quaternion spawnRot = Quaternion.identity;

GameObject gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto10x10", typeof(GameObject)), spawnPos, spawnRot);

stage = GameObject.FindWithTag("Base");

gridexpansion.transform.parent = stage.transform;

gridWidthtemp = gridWidth;

gridWidth = 10;

gridtemp = new Transform[gridWidth, gridHeight, gridWidth];

for (int y = 0; y < gridHeight; y++) {

for (int x = 0; x < gridWidthtemp; x++) {

for (int z = 0; z < gridWidthtemp; z++) {

gridtemp [x, y, z] = grid [x, y, z];

}

}

}

grid = gridtemp;

mainCam = Camera.main; *// Grab a reference to the camera*

mainCam.transform.position += new Vector3(0.5f, 0, 0);

*//The following code is in case a user uses the menu slider to go to a level manually, it ensures*

*//that the stage is expanded completely to this level.*

*//Also, it ensures the tetrominos list has all the tetrominos needed.*

if (GameObject.Find ("upgradeto4x4(Clone)") != true) {

spawnPos = new Vector3(3f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto4x4", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto5x5(Clone)") != true) {

spawnPos = new Vector3(4f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto5x5", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto6x6(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 5f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto6x6", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto7x7(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 6f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto7x7", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto8x8(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 7f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto8x8", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto9x9(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 8f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto9x9", typeof(GameObject)), spawnPos, spawnRot);

}

*//The following code ensures the tetrominos list has all the tetrominos needed.*

*//If the user decides to pick a level manually.*

if (!Tetrominos.Contains (Resources.Load ("Prefabs/DoubleCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/DoubleCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SmallLCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SmallLCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/TCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/TCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SquareCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SquareCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/ZCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/ZCube") as GameObject);

}

}

}

*// Update is called once per frame*

void Update () {

UpdateScore ();

updateTextOnUI ();

UpdateLevel ();

UpdateSpeed ();

}

*//Sync variables in this class with GUI text being displayed*

public void updateTextOnUI(){

hud\_score.text = currentScore.ToString ();

hud\_level.text = currentLevel.ToString ();

hud\_lines.text = numOfRowsCleared.ToString ();

}

*//update level*

void UpdateLevel () {

currentLevel = startingLevel + numOfRowsCleared / 1; *//level increases everytime 1 row is cleared. Divide by 10, to make the user clear 10*

*//rows before proceeding to the next level.*

*//add startinglevel to level value in order to implement slider*

ExpandGrid ();

}

void UpdateSpeed () {

fallSpeed = 1.0f - ((float)currentLevel \* 0.1f); *//speed decreases with current level.*

}

*//update score based on number of rows/lines cleared*

public void UpdateScore (){

if (numberOfRowsThisTurn > 0) {

if (numberOfRowsThisTurn == 1) {

ClearedOneRow ();

} else if (numberOfRowsThisTurn == 2) {

ClearedTwoRows ();

} else if (numberOfRowsThisTurn == 3) {

ClearedThreeRows ();

} else if (numberOfRowsThisTurn == 4) {

ClearedFourRows ();

}

numberOfRowsThisTurn = 0;

}

}

public void ClearedOneRow () { *//user cleared four rows simultaneously.*

currentScore += scoreOneLine + (currentLevel \* 20);

numOfRowsCleared++;

}

public void ClearedTwoRows () { *//user cleared four rows simultaneously.*

currentScore += scoreThreeLine + (currentLevel \* 25);

numOfRowsCleared += 2;

}

public void ClearedThreeRows () { *//user cleared four rows simultaneously.*

currentScore += scoreThreeLine + (currentLevel \* 30);

numOfRowsCleared += 3;

}

public void ClearedFourRows () { *//user cleared four rows simultaneously.*

currentScore += scoreThreeLine + (currentLevel \* 45);

numOfRowsCleared += 4;

}

*//This method will check if the game is over (i.e. the player has exceeded the maximum possible height).*

public bool CheckIsAboveGrid(Tetromino tetromino) {

foreach (Transform mino in tetromino.transform) {

Vector3 pos = Round (mino.position);

if (pos.y > gridHeight - 1) {

return true;

}

}

return false;

}

*//Checks if row is full at given y position*

public bool IsFullRowAt (int y) {

for (int x = 0; x < gridWidth; x++) {

for (int z = 0; z < gridWidth; z++) {

if (grid [x, y, z] == null) {

return false;

}

}

}

*//Found a full row, we are incrementing number of rows cleared this turn*

numberOfRowsThisTurn++;

return true;

}

public void DeleteMinoAt (int y) { *//Deletes a mino at given y position*

for (int x = 0; x < gridWidth; x++) { *//for all x positions*

for (int z = 0; z < gridWidth; z++) { *// for all z positions*

Destroy (grid [x, y, z].gameObject); *//destroy object at grid position*

grid [x, y, z] = null; *//mark position as free/null*

}

}

}

public void MoveRowDown(int y) { *//moves row down*

for (int x = 0; x < gridWidth; x++) {

for (int z = 0; z < gridWidth; z++) {

if (grid [x, y, z] != null) {

grid [x, y - 1, z] = grid [x, y, z];

grid [x, y, z] = null;

grid [x, y - 1, z].position += new Vector3 (0, -1, 0);

}

}

}

}

public void MoveAllRowDown (int y) { *//for every pos y less than gridheight, move row down*

for (int i = y; i < gridHeight; i++) {

MoveRowDown (i);

}

}

public void DeleteRow () { *//delete row*

for (int y = 0; y < gridHeight; y++) {

if (IsFullRowAt (y)) {

DeleteMinoAt (y);

MoveAllRowDown (y + 1);

y--;

}

}

}

*//This function updates grid based on which tetromino positions are taken (checks mino's X, Y and Z values)*

public void UpdateGrid (Tetromino tetromino) {

for (int y = 0; y < gridHeight; y++) {

for (int x = 0; x < gridWidth; x++) {

for (int z = 0; z < gridWidth; z++) {

if (grid [x, y, z] != null) {

if (grid [x, y, z].parent == tetromino.transform) {

grid [x, y, z] = null;

}

}

}

}

}

foreach (Transform mino in tetromino.transform) {

Vector3 pos = Round (mino.position);

if (pos.y < gridHeight) {

grid [(int) pos.x, (int) pos.y, (int) pos.z] = mino;

}

}

}

*//get mino's transform at grid position (used to check if falling tetromino is in a valid position).*

public Transform GetTransformAtGridPosition (Vector3 pos) {

if (pos.y > gridHeight - 1) {

return null;

} else {

return grid[(int) pos.x, (int) pos.y, (int) pos.z];

}

}

*//This function spawns a new tetromino piece*

public void SpawnNextTetromino(){

if (!gameStarted) {

gameStarted = true;

*// GameObject element = Tetrominos[Random.Range(0, Tetrominos.Length)]; //OLD ARRAY implementation*

GameObject element = Tetrominos[Random.Range( 0, Tetrominos.Count )];

Vector3 spawnPos = new Vector3(0f, 15.0f, 0f); *//position to spawn tetromino*

Quaternion spawnRot = Quaternion.identity; *//rotation = 0*

nextTetromino = (GameObject)Instantiate(element, spawnPos, spawnRot);

previewTetromino = (GameObject)Instantiate(element, previewTetrominoPosition, spawnRot);

previewTetromino.GetComponent<Tetromino> ().enabled = false; *//disable tetromino script because preview*

*//tetromino should not move at all*

SpawnGhostTetromino (); *//spawn ghost tetromino/piece that shows the user where the tetromino will fall*

} else {

previewTetromino.transform.localPosition = new Vector3(0f, 15.0f, 0f);

nextTetromino = previewTetromino;

nextTetromino.GetComponent<Tetromino> ().enabled = true;

*//instantiate new tetromino and set its position to false*

*// GameObject element = Tetrominos[Random.Range(0, Tetrominos.Length)]; //OLD ARRAY IMPLEMENTATION*

GameObject element = Tetrominos[Random.Range( 0, Tetrominos.Count )];

Quaternion spawnRot = Quaternion.identity;

previewTetromino = (GameObject)Instantiate(element, previewTetrominoPosition, spawnRot);

previewTetromino.GetComponent<Tetromino> ().enabled = false; *//disable tetromino script*

SpawnGhostTetromino ();

}

liveTetromino = nextTetromino;

}

public void SpawnGhostTetromino(){

if (GameObject.FindGameObjectWithTag ("CurrentGhostTetromino") != null) {

Destroy (GameObject.FindGameObjectWithTag ("CurrentGhostTetromino")); *//destroy previous ghost tetromino piece*

}

ghostTetromino = (GameObject)Instantiate (nextTetromino, new Vector3(0f, 15.0f, 0f), Quaternion.identity); *//spawn new one*

*//make ghost tetromino piece's behavior be controlled by ghosttetromino script and not tetromino script*

Destroy (ghostTetromino.GetComponent<Tetromino> ());

ghostTetromino.AddComponent<GhostTetromino> ();

}

*//function to check if tetromino is inside grid*

public bool CheckIsInsideGrid(Vector3 pos) {

stage = GameObject.FindWithTag("Stage"); *//locate stage*

return (int) pos.x >= 0 && (int) pos.x < gridWidth && (int) pos.z >= 0 && (int) pos.z < gridWidth && (int) pos.y > (int) stage.transform.position.y;

}

*//function to round positions, so we can get integers to index grid*

public Vector3 Round (Vector3 pos) {

return new Vector3 (Mathf.Round (pos.x), Mathf.Round (pos.y), Mathf.Round (pos.z));

}

*//if gameover, load gameover scene*

public void GameOver() {

SceneManager.LoadScene ("gameover");

}

}

Vector3 spawnPos = new Vector3(0f, 0f, 6f);

Quaternion spawnRot = Quaternion.identity;

GameObject gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto7x7", typeof(GameObject)), spawnPos, spawnRot);

stage = GameObject.FindWithTag("Base");

gridexpansion.transform.parent = stage.transform;

gridWidthtemp = gridWidth;

gridWidth = 7;

gridtemp = new Transform[gridWidth, gridHeight, gridWidth];

for (int y = 0; y < gridHeight; y++) {

for (int x = 0; x < gridWidthtemp; x++) {

for (int z = 0; z < gridWidthtemp; z++) {

gridtemp [x, y, z] = grid [x, y, z];

}

}

}

grid = gridtemp;

mainCam = Camera.main; *// Grab a reference to the camera\*

mainCam.transform.position += new Vector3(0.5f, 0, 0);

*//The following code is in case a user uses the menu slider to go to a level manually, it ensures*

*//that the stage is expanded completely to this level.*

*//Also, it ensures the tetrominos list has all the tetrominos needed.*

if (GameObject.Find ("upgradeto4x4(Clone)") != true) {

spawnPos = new Vector3(3f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto4x4", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto5x5(Clone)") != true) {

spawnPos = new Vector3(4f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto5x5", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto6x6(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 5f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto6x6", typeof(GameObject)), spawnPos, spawnRot);

}

*//check prev levels*

if (!Tetrominos.Contains (Resources.Load ("Prefabs/DoubleCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/DoubleCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SmallLCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SmallLCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/TCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/TCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SquareCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SquareCube") as GameObject);

}

}

else if (currentLevel == 8 && GameObject.Find("upgradeto8x8(Clone)") != true) {

Vector3 spawnPos = new Vector3(0f, 0f, 7f);

Quaternion spawnRot = Quaternion.identity;

GameObject gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto8x8", typeof(GameObject)), spawnPos, spawnRot);

stage = GameObject.FindWithTag("Base");

gridexpansion.transform.parent = stage.transform;

gridWidthtemp = gridWidth;

gridWidth = 8;

gridtemp = new Transform[gridWidth, gridHeight, gridWidth];

for (int y = 0; y < gridHeight; y++) {

for (int x = 0; x < gridWidthtemp; x++) {

for (int z = 0; z < gridWidthtemp; z++) {

gridtemp [x, y, z] = grid [x, y, z];

}

}

}

grid = gridtemp;

mainCam = Camera.main; *// Grab a reference to the camera\*

mainCam.transform.position += new Vector3(0.5f, 0, 0);

*//The following code is in case a user uses the menu slider to go to a level manually, it ensures*

*//that the stage is expanded completely to this level.*

*//Also, it ensures the tetrominos list has all the tetrominos needed.*

if (GameObject.Find ("upgradeto4x4(Clone)") != true) {

spawnPos = new Vector3(3f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto4x4", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto5x5(Clone)") != true) {

spawnPos = new Vector3(4f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto5x5", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto6x6(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 5f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto6x6", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto7x7(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 6f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto7x7", typeof(GameObject)), spawnPos, spawnRot);

}

*//The following code ensures the tetrominos list has all the tetrominos needed.*

*//If the user decides to pick a level manually.*

if (!Tetrominos.Contains (Resources.Load ("Prefabs/DoubleCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/DoubleCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SmallLCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SmallLCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/TCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/TCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SquareCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SquareCube") as GameObject);

}

}

else if (currentLevel == 9 && GameObject.Find("upgradeto9x9(Clone)") != true) {

Vector3 spawnPos = new Vector3(0f, 0f, 8f);

Quaternion spawnRot = Quaternion.identity;

GameObject gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto9x9", typeof(GameObject)), spawnPos, spawnRot);

stage = GameObject.FindWithTag("Base");

gridexpansion.transform.parent = stage.transform;

gridWidthtemp = gridWidth;

gridWidth = 9;

gridtemp = new Transform[gridWidth, gridHeight, gridWidth];

for (int y = 0; y < gridHeight; y++) {

for (int x = 0; x < gridWidthtemp; x++) {

for (int z = 0; z < gridWidthtemp; z++) {

gridtemp [x, y, z] = grid [x, y, z];

}

}

}

grid = gridtemp;

mainCam = Camera.main; *// Grab a reference to the camera\*

mainCam.transform.position += new Vector3(0.5f, 0, 0);

if (!Tetrominos.Contains (Resources.Load ("Prefabs/ZCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/ZCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SCube") as GameObject);

}

*//The following code is in case a user uses the menu slider to go to a level manually, it ensures*

*//that the stage is expanded completely to this level.*

*//Also, it ensures the tetrominos list has all the tetrominos needed.*

if (GameObject.Find ("upgradeto4x4(Clone)") != true) {

spawnPos = new Vector3(3f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto4x4", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto5x5(Clone)") != true) {

spawnPos = new Vector3(4f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto5x5", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto6x6(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 5f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto6x6", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto7x7(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 6f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto7x7", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto8x8(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 7f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto8x8", typeof(GameObject)), spawnPos, spawnRot);

}

*//The following code ensures the tetrominos list has all the tetrominos needed.*

*//If the user decides to pick a level manually.*

if (!Tetrominos.Contains (Resources.Load ("Prefabs/DoubleCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/DoubleCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SmallLCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SmallLCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/TCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/TCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SquareCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SquareCube") as GameObject);

}

}

else if (currentLevel == 10 && GameObject.Find("upgradeto10x10(Clone)") != true) {

Vector3 spawnPos = new Vector3(0f, 0f, 9f);

Quaternion spawnRot = Quaternion.identity;

GameObject gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto10x10", typeof(GameObject)), spawnPos, spawnRot);

stage = GameObject.FindWithTag("Base");

gridexpansion.transform.parent = stage.transform;

gridWidthtemp = gridWidth;

gridWidth = 10;

gridtemp = new Transform[gridWidth, gridHeight, gridWidth];

for (int y = 0; y < gridHeight; y++) {

for (int x = 0; x < gridWidthtemp; x++) {

for (int z = 0; z < gridWidthtemp; z++) {

gridtemp [x, y, z] = grid [x, y, z];

}

}

}

grid = gridtemp;

mainCam = Camera.main; *// Grab a reference to the camera*

mainCam.transform.position += new Vector3(0.5f, 0, 0);

*//The following code is in case a user uses the menu slider to go to a level manually, it ensures*

*//that the stage is expanded completely to this level.*

*//Also, it ensures the tetrominos list has all the tetrominos needed.*

if (GameObject.Find ("upgradeto4x4(Clone)") != true) {

spawnPos = new Vector3(3f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto4x4", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto5x5(Clone)") != true) {

spawnPos = new Vector3(4f, 0f, 0f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto5x5", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto6x6(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 5f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto6x6", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto7x7(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 6f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto7x7", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto8x8(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 7f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto8x8", typeof(GameObject)), spawnPos, spawnRot);

}

if (GameObject.Find ("upgradeto9x9(Clone)") != true) {

spawnPos = new Vector3(0f, 0f, 8f); *//where to spawn new grid prefab to fit in with preexisting grid*

spawnRot = Quaternion.identity; *//rotation = 0, 0, 0*

*//instantiate object*

gridexpansion = (GameObject)Instantiate(Resources.Load ("Prefabs/upgradeto9x9", typeof(GameObject)), spawnPos, spawnRot);

}

*//The following code ensures the tetrominos list has all the tetrominos needed.*

*//If the user decides to pick a level manually.*

if (!Tetrominos.Contains (Resources.Load ("Prefabs/DoubleCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/DoubleCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SmallLCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SmallLCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/TCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/TCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SquareCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SquareCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/SCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/SCube") as GameObject);

}

if (!Tetrominos.Contains (Resources.Load ("Prefabs/ZCube") as GameObject)) {

Tetrominos.Add (Resources.Load ("Prefabs/ZCube") as GameObject);

}

}

}

*// Update is called once per frame*

void Update () {

UpdateScore ();

updateTextOnUI ();

UpdateLevel ();

UpdateSpeed ();

}

*//Sync variables in this class with GUI text being displayed*

public void updateTextOnUI(){

hud\_score.text = currentScore.ToString ();

hud\_level.text = currentLevel.ToString ();

hud\_lines.text = numOfRowsCleared.ToString ();

}

*//update level*

void UpdateLevel () {

currentLevel = startingLevel + numOfRowsCleared / 1; *//level increases everytime 1 row is cleared. Divide by 10, to make the user clear 10*

*//rows before proceeding to the next level.*

*//add startinglevel to level value in order to implement slider*

ExpandGrid ();

}

void UpdateSpeed () {

fallSpeed = 1.0f - ((float)currentLevel \* 0.1f); *//speed decreases with current level.*

}

*//update score based on number of rows/lines cleared*

public void UpdateScore (){

if (numberOfRowsThisTurn > 0) {

if (numberOfRowsThisTurn == 1) {

ClearedOneRow ();

} else if (numberOfRowsThisTurn == 2) {

ClearedTwoRows ();

} else if (numberOfRowsThisTurn == 3) {

ClearedThreeRows ();

} else if (numberOfRowsThisTurn == 4) {

ClearedFourRows ();

}

numberOfRowsThisTurn = 0;

}

}

public void ClearedOneRow () { *//user cleared four rows simultaneously.*

currentScore += scoreOneLine + (currentLevel \* 20);

numOfRowsCleared++;

}

public void ClearedTwoRows () { *//user cleared four rows simultaneously.*

currentScore += scoreThreeLine + (currentLevel \* 25);

numOfRowsCleared += 2;

}

public void ClearedThreeRows () { *//user cleared four rows simultaneously.*

currentScore += scoreThreeLine + (currentLevel \* 30);

numOfRowsCleared += 3;

}

public void ClearedFourRows () { *//user cleared four rows simultaneously.*

currentScore += scoreThreeLine + (currentLevel \* 45);

numOfRowsCleared += 4;

}

*//This method will check if the game is over (i.e. the player has exceeded the maximum possible height).*

public bool CheckIsAboveGrid(Tetromino tetromino) {

foreach (Transform mino in tetromino.transform) {

Vector3 pos = Round (mino.position);

if (pos.y > gridHeight - 1) {

return true;

}

}

return false;

}

*//Checks if row is full at given y position*

public bool IsFullRowAt (int y) {

for (int x = 0; x < gridWidth; x++) {

for (int z = 0; z < gridWidth; z++) {

if (grid [x, y, z] == null) {

return false;

}

}

}

*//Found a full row, we are incrementing number of rows cleared this turn*

numberOfRowsThisTurn++;

return true;

}

public void DeleteMinoAt (int y) { *//Deletes a mino at given y position*

for (int x = 0; x < gridWidth; x++) { *//for all x positions*

for (int z = 0; z < gridWidth; z++) { *// for all z positions*

Destroy (grid [x, y, z].gameObject); *//destroy object at grid position*

grid [x, y, z] = null; *//mark position as free/null*

}

}

}

public void MoveRowDown(int y) { *//moves row down*

for (int x = 0; x < gridWidth; x++) {

for (int z = 0; z < gridWidth; z++) {

if (grid [x, y, z] != null) {

grid [x, y - 1, z] = grid [x, y, z];

grid [x, y, z] = null;

grid [x, y - 1, z].position += new Vector3 (0, -1, 0);

}

}

}

}

public void MoveAllRowDown (int y) { *//for every pos y less than gridheight, move row down*

for (int i = y; i < gridHeight; i++) {

MoveRowDown (i);

}

}

public void DeleteRow () { *//delete row*

for (int y = 0; y < gridHeight; y++) {

if (IsFullRowAt (y)) {

DeleteMinoAt (y);

MoveAllRowDown (y + 1);

y--;

}

}

}

*//This function updates grid based on which tetromino positions are taken (checks mino's X, Y and Z values)*

public void UpdateGrid (Tetromino tetromino) {

for (int y = 0; y < gridHeight; y++) {

for (int x = 0; x < gridWidth; x++) {

for (int z = 0; z < gridWidth; z++) {

if (grid [x, y, z] != null) {

if (grid [x, y, z].parent == tetromino.transform) {

grid [x, y, z] = null;

}

}

}

}

}

foreach (Transform mino in tetromino.transform) {

Vector3 pos = Round (mino.position);

if (pos.y < gridHeight) {

grid [(int) pos.x, (int) pos.y, (int) pos.z] = mino;

}

}

}

*//get mino's transform at grid position (used to check if falling tetromino is in a valid position).*

public Transform GetTransformAtGridPosition (Vector3 pos) {

if (pos.y > gridHeight - 1) {

return null;

} else {

return grid[(int) pos.x, (int) pos.y, (int) pos.z];

}

}

*//This function spawns a new tetromino piece*

public void SpawnNextTetromino(){

if (!gameStarted) {

gameStarted = true;

*// GameObject element = Tetrominos[Random.Range(0, Tetrominos.Length)]; //OLD ARRAY implementation*

GameObject element = Tetrominos[Random.Range( 0, Tetrominos.Count )];

Vector3 spawnPos = new Vector3(0f, 15.0f, 0f); *//position to spawn tetromino*

Quaternion spawnRot = Quaternion.identity; *//rotation = 0*

nextTetromino = (GameObject)Instantiate(element, spawnPos, spawnRot);

previewTetromino = (GameObject)Instantiate(element, previewTetrominoPosition, spawnRot);

previewTetromino.GetComponent<Tetromino> ().enabled = false; *//disable tetromino script because preview*

*//tetromino should not move at all*

SpawnGhostTetromino (); *//spawn ghost tetromino/piece that shows the user where the tetromino will fall*

} else {

previewTetromino.transform.localPosition = new Vector3(0f, 15.0f, 0f);

nextTetromino = previewTetromino;

nextTetromino.GetComponent<Tetromino> ().enabled = true;

*//instantiate new tetromino and set its position to false*

*// GameObject element = Tetrominos[Random.Range(0, Tetrominos.Length)]; //OLD ARRAY IMPLEMENTATION*

GameObject element = Tetrominos[Random.Range( 0, Tetrominos.Count )];

Quaternion spawnRot = Quaternion.identity;

previewTetromino = (GameObject)Instantiate(element, previewTetrominoPosition, spawnRot);

previewTetromino.GetComponent<Tetromino> ().enabled = false; *//disable tetromino script*

SpawnGhostTetromino ();

}

liveTetromino = nextTetromino;

}

public void SpawnGhostTetromino(){

if (GameObject.FindGameObjectWithTag ("CurrentGhostTetromino") != null) {

Destroy (GameObject.FindGameObjectWithTag ("CurrentGhostTetromino")); *//destroy previous ghost tetromino piece*

}

ghostTetromino = (GameObject)Instantiate (nextTetromino, new Vector3(0f, 15.0f, 0f), Quaternion.identity); *//spawn new one*

*//make ghost tetromino piece's behavior be controlled by ghosttetromino script and not tetromino script*

Destroy (ghostTetromino.GetComponent<Tetromino> ());

ghostTetromino.AddComponent<GhostTetromino> ();

}

*//function to check if tetromino is inside grid*

public bool CheckIsInsideGrid(Vector3 pos) {

stage = GameObject.FindWithTag("Stage"); *//locate stage*

return (int) pos.x >= 0 && (int) pos.x < gridWidth && (int) pos.z >= 0 && (int) pos.z < gridWidth && (int) pos.y > (int) stage.transform.position.y;

}

*//function to round positions, so we can get integers to index grid*

public Vector3 Round (Vector3 pos) {

return new Vector3 (Mathf.Round (pos.x), Mathf.Round (pos.y), Mathf.Round (pos.z));

}

*//if gameover, load gameover scene*

public void GameOver() {

SceneManager.LoadScene ("gameover");

}

}

**Week 4: Week of June 26th**

*Day 1: June 26th:*

Modified the code for the other aspects of the game a little bit. The code for the base of the game Levels 1 to 10 was written by Friday of last week. However, the other aspects of the game such as the code for the buttons to move the tetromino in the positive/negative x, y, and z planes, and the code for the gamemenu had to be modified or rewritten:

Code for buttons:

using UnityEngine;

using System.Collections;

public class Buttons : MonoBehaviour {

*// Use this for initialization*

void Start () {

}

*// Update is called once per frame*

void Update () {

}

void OnMouseDown() {

Tetromino tetro = FindObjectOfType<Game> ().liveTetromino.GetComponent<Tetromino>(); *//get tetromino*

switch (name) {

case "Move\_X\_pos":

tetro.MoveRight ();

break;

case "Move\_X\_neg":

tetro.MoveLeft ();

break;

case "Move\_Z\_pos":

tetro.MoveIn ();

break;

case "Move\_Z\_neg":

tetro.MoveOut ();

break;

case "Rotate\_X":

tetro.RotateX ();

break;

case "Rotate\_Y":

tetro.RotateY ();

break;

case "Rotate\_Z":

tetro.RotateZ ();

break;

}

*//don't need y key because user cant move down (will add button later)*

tetro.KeyReleasedHorizontal (); *//user released key*

tetro.KeyReleasedZ (); *//user released key*

}

}

Code for GameMenu:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement; *//in order to load scenemanager*

using UnityEngine.UI; *//to update text*

public class GameMenu : MonoBehaviour {

public Text levelText;

*// Use this for initialization*

void Start () {

levelText.text = "0";

}

public void PlayGame () {

if (Game.startingLevel == 0) {

Game.startingAtLevelZero = true;

} else {

Game.startingAtLevelZero = false;

}

SceneManager.LoadScene ("level1");

}

public void ChangedLevel (float value) {

Game.startingLevel = (int)value;

levelText.text = value.ToString ();

}

public void LaunchGameMenu(){

SceneManager.LoadScene ("gamemenu");

}

}

*Day 2: June 27th:*

Modified the code for the tetrominoes and cleaned it up a bit. Added comments to make it clearer when we have to go back and fix it some more.

*//Activates triggers*

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Tetromino : MonoBehaviour {

*//variable used to make the tetromino piece fall down automatically*

float fall = 0;

private float fallSpeed; *//the smaller the value, the faster the piece will fall. It decreases with number of rows cleared.*

*//These variables can be used to limit rotation or disallow certain objects from rotating. (Not in use).*

public bool allowRotation = true;

public bool limitRotation = false;

private GameObject stage; *//this variable is used to identify where the stage is*

private Camera mainCam; *//not in use*

*//score variables*

public int individualScore = 100; *//100 represents what the score is going to be if the player immediately sets piece down*

private float individualScoreTime; *//time it takes for player to place piece down*

*//score end*

*//variables used to allow player to hold movement buttons and not have to keep pressing and releasing them*

private float continuousVerticalSpeed = 0.05f; *//the speed at which the tetromino will move when the down button is held down*

private float continuousZSpeed = 0.1f;

private float continuousHorizontalSpeed = 0.1f; *//speed at which the tetromino will move when the left/right buttons are held down*

private float buttonDownWaitMax = 0.2f; *//how long to wait before the tetromino recognizes button is being held down*

private float verticalTimer = 0f;

private float horizontalTimer = 0f;

private float ZTimer = 0f;

private float buttonDownWaitTimerHorizontal = 0f;

private float buttonDownWaitTimerVertical = 0f;

private float buttonDownWaitTimerZ = 0f;

private bool movedImmediateHorizontal = false;

private bool movedImmediateVertical = false;

private bool movedImmediateZ = false;

*//end of movement variables*

*//Touch movement variables*

private int touchSensitivityHorizontal = 10;

private int touchSensitivityVertical = 2;

Vector2 previousUnitPosition = Vector2.zero;

Vector2 direction = Vector2.zero;

bool moved = false;

*//end of touch movement variables*

void Start () {

mainCam = Camera.main; *// Grab a reference to the camera (Not in use).*

fallSpeed = FindObjectOfType<Game> ().fallSpeed; *//finds fall speed of tetromino piece from game class*

}

*// Update is called once per frame*

void Update () {

CheckUserInput (); *//every frame checks for button user presses*

UpdateIndividualScore (); *//updates score*

}

void UpdateIndividualScore(){

if (individualScoreTime < 1) {

individualScoreTime += Time.deltaTime;

} else {

individualScoreTime = 0;

individualScore = Mathf.Max (individualScore - 10, 0); *//Mathf.Max so that score cannot drop below 0*

}

}

void CheckUserInput(){ *//checks what button the user pressed*

*//TOUCH Implementation*

if (Input.touchCount > 0) {

Touch t = Input.GetTouch (0);

if (t.phase == TouchPhase.Began) {

previousUnitPosition = new Vector2 (t.position.x, t.position.y);

} else if (t.phase == TouchPhase.Moved) {

Vector2 touchDeltaPosition = t.deltaPosition;

direction = touchDeltaPosition.normalized;

if (Mathf.Abs (t.position.x - previousUnitPosition.x) <= touchSensitivityHorizontal) {

if (Mathf.Abs (t.position.y - previousUnitPosition.y) >= touchSensitivityVertical) {

if (direction.y < 0) {

MoveDown ();

previousUnitPosition = t.position;

moved = true;

}

}

}

} else if (t.phase == TouchPhase.Ended) {

moved = false;

}

}

if (Time.time - fall >= fallSpeed){

MoveDown();

}

if (Input.GetKeyUp (KeyCode.LeftArrow) || Input.GetKeyUp (KeyCode.RightArrow)) {

KeyReleasedHorizontal (); *//user released key, reset movement variables*

}

if (Input.GetKeyUp (KeyCode.S)) {

KeyReleasedVertical (); *//user released key, reset movement variables*

}

if (Input.GetKeyUp (KeyCode.DownArrow) || Input.GetKeyUp (KeyCode.UpArrow)) {

KeyReleasedZ (); *//user released key, reset movement variables*

}

*//if user presses/hold S, the piece will move down faster, but either way, it moves down at a constant speed*

if (Input.GetKey (KeyCode.S) || Time.time - fall >= fallSpeed) {

MoveDown ();

}

*//if user presses/holds left arrow, the piece will move left*

if (Input.GetKey (KeyCode.LeftArrow)) {

MoveLeft ();

} else if (Input.GetKey (KeyCode.RightArrow)) { *//if user presses/holds right arrow, the piece will move right*

MoveRight ();

}

if (Input.GetKey (KeyCode.DownArrow)) { *//if user presses/holds down arrow, the piece will towards Z negative*

MoveOut ();

}

else if (Input.GetKey (KeyCode.UpArrow)) { *//if user presses/holds up arrow, the piece will towards Z positive*

MoveIn ();

} if (Input.GetKeyDown (KeyCode.Space)) { *//if user presses space, the tetromino will rotate 90 degrees in the Z direction*

RotateZ ();

}

*// Rotation along X and Y*

if (Input.GetKeyDown (KeyCode.X)) { *//if user presses space, the tetromino will rotate 90 degrees in the X direction*

RotateX ();

}

if (Input.GetKeyDown (KeyCode.Z)) { *//if user presses space, the tetromino will rotate 90 degrees in the Y direction*

RotateY ();

}

}

*//Moves piece to the left*

public void MoveLeft(){

*//The code below allows the user to hold down movement button and move piece 1 unit at a time.*

*//It works by resetting a timer when the user releases the button.*

if (movedImmediateHorizontal) {

if (buttonDownWaitTimerHorizontal < buttonDownWaitMax) {

buttonDownWaitTimerHorizontal += Time.deltaTime;

return;

}

if (horizontalTimer < continuousHorizontalSpeed) {

horizontalTimer += Time.deltaTime;

return;

}

}

if (!movedImmediateHorizontal) {

movedImmediateHorizontal = true;

}

horizontalTimer = 0;

*//end of movement code*

transform.position += new Vector3 (-1, 0, 0); *//subtracts one unit from the tetromino's X position*

if (CheckIsValidPosition ()) { *//checks if position is valid*

FindObjectOfType<Game> ().UpdateGrid (this); *//if valid, updates grid system so that its position becomes unavailable*

} else {

transform.position += new Vector3 (1, 0, 0); *//if not valid, undo movement (happens very quickly so user does not notice).*

}

}

*//Moves piece to the right*

public void MoveRight(){

*//The code below allows the user to hold down movement button and move piece 1 unit at a time.*

*//It works by resetting a timer when the user releases the button.*

if (movedImmediateHorizontal) {

if (buttonDownWaitTimerHorizontal < buttonDownWaitMax) {

buttonDownWaitTimerHorizontal += Time.deltaTime;

return;

}

if (horizontalTimer < continuousHorizontalSpeed) {

horizontalTimer += Time.deltaTime;

return;

}

}

if (!movedImmediateHorizontal) {

movedImmediateHorizontal = true;

}

horizontalTimer = 0;

*//end of movement code*

transform.position += new Vector3 (1, 0, 0); *//adds one unit to the tetromino's X position*

if (CheckIsValidPosition ()) {

FindObjectOfType<Game> ().UpdateGrid (this);

} else {

transform.position += new Vector3 (-1, 0, 0);

}

}

Also began to work on the WIP #1 presentation for thursday. I created the powerpoint and filled it with the overview, objective, background information, changes made/progress, to-do/future slides. I asked Eugene to work on adding pictures from each level so that it would be clearer to the audience and they can see exactly what’s happening at each level. Eugene is also going to work on previous research and talking about existing platforms like Chemtris.

*Day 3: June 28th:*

Continued modifying the code for the tetrominoes:

*//Moves piece down*

public void MoveDown(){

*//The code below allows the user to hold down movement button and move piece 1 unit at a time.*

*//It works by resetting a timer when the user releases the button.*

if (movedImmediateVertical) {

if (buttonDownWaitTimerVertical < buttonDownWaitMax) {

buttonDownWaitTimerVertical += Time.deltaTime;

return;

}

if (verticalTimer < continuousVerticalSpeed) {

verticalTimer += Time.deltaTime;

return;

}

}

if (!movedImmediateVertical) {

movedImmediateVertical = true;

}

verticalTimer = 0;

*//end of movement code*

transform.position += new Vector3 (0, -1, 0); *//subtracts one unit from the tetromino's Y position*

fall = Time.time; *//update fall variable so that tetromino continues to fall automatically*

if (CheckIsValidPosition ()) {

FindObjectOfType<Game> ().UpdateGrid (this);

} else { *//if position is not valid*

transform.position += new Vector3 (0, 1, 0);

*//disable tetromino because it is no longer inside the grid (i.e. it fell onto the grid or a bug occurred).*

enabled = false;

*//clear tag so that new ghost tetromino spawns and follows the NEW tetromino and not the old tagged one.*

tag = "Untagged";

*//check if row is full after new piece has landed*

FindObjectOfType<Game> ().DeleteRow ();

if (FindObjectOfType<Game> ().CheckIsAboveGrid (this)) {

*//if tetromino piece is above max height of grid, then game ends*

FindObjectOfType<Game> ().GameOver();

}

*//stage = GameObject.Find ("Stage (14)"); THIS IS AN ALTERNATE WAY OF FINDING STAGE (more expensive however).*

*//make child of stage, so as to stick to make the tetromino stick to the stage, in case we decide to move the stage*

stage = GameObject.FindWithTag("Stage");

transform.parent = stage.transform;

*//spawn new tetromino piece*

FindObjectOfType<Game> ().SpawnNextTetromino ();

*//update score as piece has landed onto grid*

Game.currentScore += individualScore;

}

}

*//Moves piece in the Z+ direction*

public void MoveIn(){

*//The code below allows the user to hold down movement button and move piece 1 unit at a time.*

*//It works by resetting a timer when the user releases the button.*

if (movedImmediateZ) {

if (buttonDownWaitTimerZ < buttonDownWaitMax) {

buttonDownWaitTimerZ += Time.deltaTime;

return;

}

if (ZTimer < continuousZSpeed) {

ZTimer += Time.deltaTime;

return;

}

}

if (!movedImmediateZ) {

movedImmediateZ = true;

}

ZTimer = 0;

*//end of movement code*

transform.position += new Vector3 (0, 0, 1); *//Add 1 to the tetromino's Z position*

if (CheckIsValidPosition ()) {

FindObjectOfType<Game> ().UpdateGrid (this);

} else {

transform.position += new Vector3 (0, 0, -1); *//Subtract 1 from the tetromino's Z position*

}

}

*//Moves piece in the Z- direction*

public void MoveOut(){

*//The code below allows the user to hold down movement button and move piece 1 unit at a time.*

*//It works by resetting a timer when the user releases the button.*

if (movedImmediateZ) {

if (buttonDownWaitTimerZ < buttonDownWaitMax) {

buttonDownWaitTimerZ += Time.deltaTime;

return;

}

if (ZTimer < continuousZSpeed) {

ZTimer += Time.deltaTime;

return;

}

}

if (!movedImmediateZ) {

movedImmediateZ = true;

}

ZTimer = 0;

*//end of movement code*

transform.position += new Vector3 (0, 0, -1);

if (CheckIsValidPosition ()) {

FindObjectOfType<Game> ().UpdateGrid (this);

} else {

transform.position += new Vector3 (0, 0, 1);

}

}

*//Rotates the piece 90 degrees on the X axis*

public void RotateX(){

transform.Rotate (90, 0, 0, Space.World);

if (CheckIsValidPosition ()) {

FindObjectOfType<Game> ().UpdateGrid (this);

} else {

transform.Rotate (-90, 0, 0, Space.World);

}

}

*//Rotates the piece 90 degrees on the Y axis*

public void RotateY(){

transform.Rotate (0, 90, 0, Space.World);

if (CheckIsValidPosition ()) {

FindObjectOfType<Game> ().UpdateGrid (this);

} else {

transform.Rotate (0, -90, 0, Space.World);

}

}

*//Rotates the piece 90 degrees on the Z axis*

public void RotateZ(){

transform.Rotate (0, 0, 90, Space.World);

if (CheckIsValidPosition ()) {

FindObjectOfType<Game> ().UpdateGrid (this);

} else {

transform.Rotate (0, 0, -90, Space.World);

}

}

*//Functions that reset movement variables after user releases key.*

public void KeyReleasedHorizontal () {

movedImmediateHorizontal = false;

horizontalTimer = 0;

buttonDownWaitTimerHorizontal = 0;

}

public void KeyReleasedVertical () {

movedImmediateVertical = false;

verticalTimer = 0;

buttonDownWaitTimerVertical = 0;

}

public void KeyReleasedZ () {

movedImmediateZ = false;

ZTimer = 0;

buttonDownWaitTimerZ = 0;

}

*//end of movement variable functions*

*//This function checks if every mino (single cube that a tetromino is made out of) in a tetromino is in a valid position*

*//i.e. it is in the grid, or above/below a certain x, y val*

bool CheckIsValidPosition () {

foreach (Transform mino in transform) { *//get every child of current tetromino piece (all the minos)*

Vector3 pos = FindObjectOfType<Game> ().Round (mino.position); *//use round function from game class to round position*

if (FindObjectOfType<Game> ().CheckIsInsideGrid (pos) == false) { *//check if inside grid*

return false; *//if not, return false*

}

if (FindObjectOfType<Game> ().GetTransformAtGridPosition (pos) != null && FindObjectOfType<Game> ().GetTransformAtGridPosition (pos).parent != transform) {

return false;

}

}

return true; *//position valid*

}

}

Also began looking for articles for this week’s article reviews. It is somewhat hard to find relevant articles now because I found articles already about game design and about educational computational molecular modeling. Other than those two, Chemtris doesn't really fall into any other categories. I tried looking for articles about organic chemistry or biochemistry and tried to relate it back to how Chemtris would help in making those concepts easier to understand. Google Scholar wasnt really helpful anymore so I started looking for articles using the Bern Dibner LIbrary Database, SpringerLInk, and SciFinder Scholar

*Day 4: June 29th*

This morning we had our In-Lab Photography sessions and following that, I had an interview with Abhi about my experiences as a woman in STEM. I forced myself to get out of my comfort zone and talk in front of a camera for this interview because I felt that it was a very important topic and if I’m given the opportunity to share both my positive/negative experiences as a woman in STEM and especially share some advice for other women in STEM, I should take advantage of it and I’m glad I did. It helped me to reflect and gain a better perspective as to why I’m in this field and what my end goal is.

Eugene and I continued to work on our powerpoint for the first Work in Progress the rest of the day. We added on some more slides about the methods used which were the two computer softwares (LibGDX and Unity) used to make and modify Chemtris. We also added pictures of the Levels 1-10 and a video of the game in motion.

The ending part of the day was BMS presentations and I found the Staff of Gandalf project to be the most interesting. I was intrigued by the number of changes and different prototypes they’ve been able to work on and I also love that three different students from completely different schools and backgrounds are able to collaborate and work on a project that can revolutionize the way the blind and visually impaired go about their day to day lives.

I also really liked the Dead Man Walking project because I think it will have a lot of real-world benefits and I can’t wait to see how they make progresses with it over the course of the next few weeks.

*Day 5: June 30th*

I first had my student headshot taken at 9:00am which took 5 minutes. I spent the rest of the day working on some design ideas for the game. At the WIP #1, Professor Lee said that we should work on changing the colors and making them brighter as well as modifying the background and adding background music and sound effects. So Eugene and I worked on a google doc which we shared with Mohamed containing all the possible design ideas regarding color and music that we have for the game and how to implement them.

I worked on both my article reviews for the week. I found the first article Computerized Molecular Modeling as a Tool To Improve Chemistry Teaching by Nitza Barnea and Yehudit J. Dori in which they test the effectiveness of computerized molecular models (CMM) in an educational setting by setting up a 14 hour workshop in which CMM was incorporated into a pre and in-service educational training in the Department of Education in Technology and Science at the Technion. Most teachers had a favorable reaction to CMM but were hesitant to actually use it in the classroom due to an anxiety to work with computers/insufficient access to technology. They also tested CMM in three experimental 10th grade classes with two control classes and the results showed that the experimental group scored higher on tests that tested the students’ understanding of concepts related to chemical structure and bonding as well as geometric and symbolic representation.

**Week 5: Week of July 3rd**

*Day 1: July 3rd:*

I spent most of today working on implementing the sound effects for the game. Unity has a tutorial on how to add sound effects to your game that are triggered by game events, using C# scripting by also reviewing the main audio components of Unity: Audio Listeners, Audio Sources and Audio Clips. The code for the ThrowObject to make a sound written in C# is:

using UnityEngine;  
using System.Collections;  
  
public class ThrowObject : MonoBehaviour {  
  
 public GameObject projectile;  
 public AudioClip shootSound;  
  
  
 private float throwSpeed = 2000f;  
 private AudioSource source;  
 private float volLowRange = .5f;  
 private float volHighRange = 1.0f;  
  
  
 void Awake () {  
   
 source = GetComponent<AudioSource>();  
  
 }  
  
  
 void Update () {  
  
 if ([Input.GetButtonDown](http://docs.unity3d.com/Documentation/ScriptReference/Input.GetButtonDown.html)("Fire1"))  
 {  
 float vol = Random.Range (volLowRange, volHighRange);  
 source.PlayOneShot(shootSound,vol);  
 GameObject throwThis = Instantiate (projectile, transform.position, transform.rotation) as GameObject;  
 throwThis.rigidbody.AddRelativeForce (new Vector3(0,0,throwSpeed));  
 }  
   
 }  
}

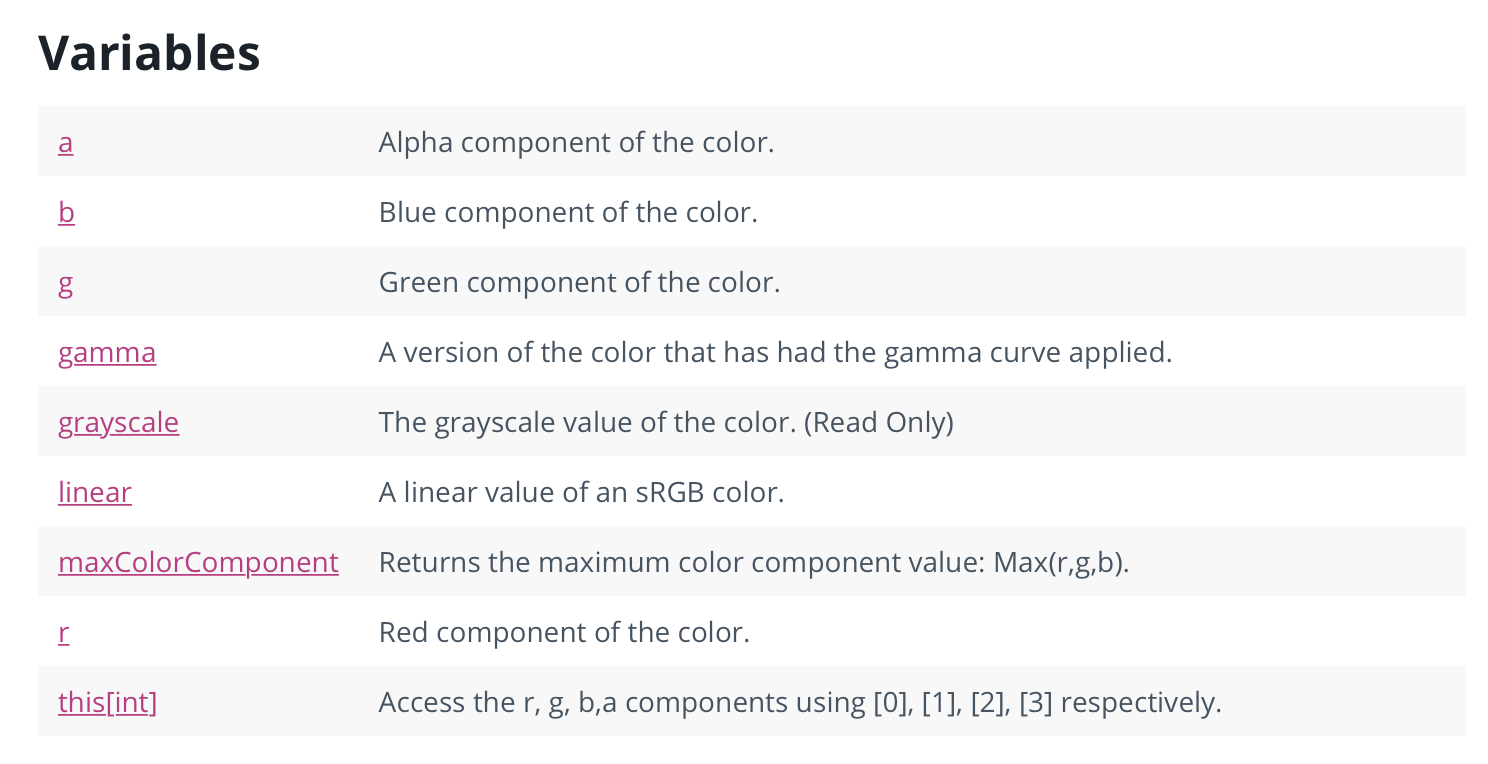
The code written for a CrashSound written in C# is:

using UnityEngine;  
using System.Collections;  
  
public class CrashSound : MonoBehaviour {  
  
 public AudioClip crashSoft;  
 public AudioClip crashHard;  
  
  
 private AudioSource source;  
 private float lowPitchRange = .75F;  
 private float highPitchRange = 1.5F;  
 private float velToVol = .2F;  
 private float velocityClipSplit = 10F;  
  
  
 void Awake () {  
   
 source = GetComponent<AudioSource>();  
 }  
  
  
 void OnCollisionEnter (Collision coll)  
 {  
 source.pitch = Random.Range (lowPitchRange,highPitchRange);  
 float hitVol = coll.relativeVelocity.magnitude \* velToVol;  
 if (coll.relativeVelocity.magnitude < velocityClipSplit)  
 source.PlayOneShot(crashSoft,hitVol);  
 else   
 source.PlayOneShot(crashHard,hitVol);  
 }  
  
}

*Day 2: July 5th:*

I spent today working on changing the colors to bright Candy Crush like colors that make students want to keep playing Chemtris and keep learning. Because the concept of adding colors to games is very subjective and difficult to grasp, I turned to the Unity website to get a little bit of help regarding Scripting API. I found that the structure used throughout Unity to pass colors around is given in a nice, easy to read chart. Each color component is a floating point value with a range from 0 to 1. Components ([r](https://docs.unity3d.com/ScriptReference/Color-r.html),[g](https://docs.unity3d.com/ScriptReference/Color-g.html),[b](https://docs.unity3d.com/ScriptReference/Color-b.html)) define a color in RGB color space. Alpha component ([a](https://docs.unity3d.com/ScriptReference/Color-a.html)) defines transparency - alpha of one is completely opaque, alpha of zero is completely transparent.





Also, the code for scripting the alpha component of the color is:

using UnityEngine;  
using System.Collections;

public class ExampleClass : [MonoBehaviour](https://docs.unity3d.com/ScriptReference/MonoBehaviour.html) {  
 public [Color](https://docs.unity3d.com/ScriptReference/Color.html) color = [Color.white](https://docs.unity3d.com/ScriptReference/Color-white.html);  
 void Example() {  
 color.a = 0;  
 }  
}

*Day 3: July 6th:*

I spent today mostly reading through various articles trying to find previous research done on the optimal colors in instructional tools or educational game design. I found the article Intelligent for Gaming Environments to be extremely useful because the author Magy Seif El-Nasr of Pennsylvania State University discusses the use of game lighting and the importance of proper lighting for a game’s overall feel. Lighting design has many functions in game environments such as directing the player’s attention, setting the ideal atmosphere, provoking emotions from the player, and providing good depth and action visibility. She talks about the pros and cons of existing lighting softwares and why using a new lighting engine called ELE or Expressive Lighting Engine in games would be useful. ELE is a lighting system that sets and adjusts the scene’s lighting in real time to allow for the optimal aesthetic and communicative functions of the game while also achieving all the goals of good lighting for a game such as evoking emotion and directing the user’s focus. The results of the article show that using ELE as a separate system that interacts with the graphic engine through a standard interface have included the improvement of the overall user experience as well as an acceleration in the development of the game because ELE provided the game with an automatic lighting system removing the need for designers to change the lighting parameters at each scene and level. I found this article to be extremely useful because it gave me the background knowledge I needed in order to make good decisions about lighting and camera position in the game design. It also gave me the idea of possibly using ELE as a separate system that interacts with the game so that it would automatically modify the lighting in the game according to real-time and not require me or Eugene to manually modify the lighting and camera position for each scene and each level.

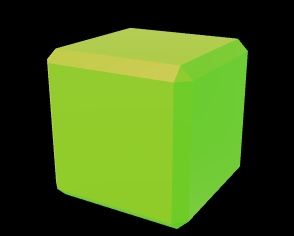
I also found an article called Color Research and its Application to the Design of Instructional Materials by Dennis Pett and Trudy Wilson in which they talk about the concepts of the physiological and psychological effects of color as well as the use of color to place players in an intrigued mindset where they want to continue learning. They go into extreme detail on how color is related to acuity, legibility, and learning as well as how color can induce certain emotions when used correctly. I found it to be really helpful because one of the major points of feedback we got from the first WIP was to modify the colors of the tetrominoes of the game so that we could make the game very interesting and fun to play.

*Day 4: July 7th:*

I continued to play around with the colors for a majority of the day. I found it somewhat difficult because the code that is complete isn't the same code as the code that is corresponding to the newest version of the game so it’s a little hard to keep track of what each person is doing and how we’re all contributing to the game because we can't visually see the changes in the updated version. I tried to manually change the color of each tetromino to make them more aesthetically pleasing and brighter.

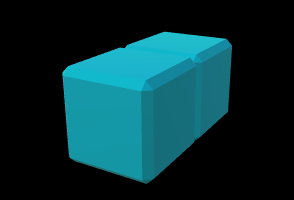
single cube (1x1x1):

* neon green
* Albedo:
  + R: 255
  + G: 191
  + B: 0
  + A: 255
* Emission Color:
  + R: 0
  + G: 0.603
  + B: 0.0852



double tetromino (1x2x1):

* neon blue
* Albedo:
  + R: 0
  + G: 255
  + B: 255
  + A: 255
* Emission Color:
  + R: 0
  + G: 0.7034
  + B: 0

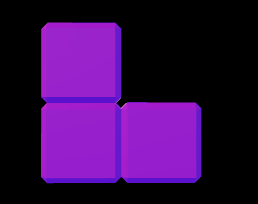


*Day 5: July 8th:*

I continued working on the colors for the small L cube and the plus sign (T-cube). This required a lot more time and effort because there was only a limited range of colors to use and the Unity settings only allowed for me to modify the red, green, and blue values for the albedo and the emission color categories without making the tetrominoes too dark to see but also bright enough to be attractive and eye-catching.

small L cube (1x2x1 with a 1x1x1 cube attached):

* neon purple
* Albedo:
  + R: 255
  + G: 0
  + B: 255
  + A: 255
* Emission Color:
  + R: 0
  + G: 0
  + B: 1



Plus sign (T-cube):

* neon pink/red
* Albedo:
  + R: 255
  + G: 128
  + B: 128
  + A: 255
* Emission Color:
  + R: 1
  + G: 0
  + B: 0



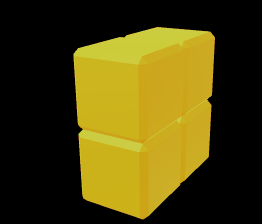
**Week 6: Week of July 10th**

*Day 1: July 10th:*

The next installment of colors were modified, which were the donut piece (square shaped cube), and the Z/S shapes.

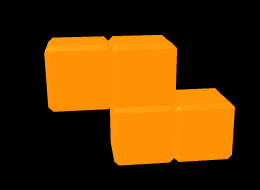
Donut (Square):

* neon yellow
* Albedo:
  + R: 255
  + G: 76
  + B: 0
  + A: 255
* Emission Color:
  + R: 1
  + G: 0.6412
  + B: 0



Z cube and S cube:

* neon orange
* Albedo:
  + R: 255
  + G: 0
  + B: 0
  + A: 255
* Emission Color:
  + R: 1
  + G: 0.5586
  + B: 0

**

*Day 2: July 11th:*

I started doing my article reviews for the week. I found an article by Carol Koroghlanian of University of Wisconsin and James D. Klein of Arizona State University called The Effect of Audio and Animation in Multimedia Instruction. In this article, both of them conduct a study in which they investigate the effects of audio, animation, and spatial ability in a multimedia computer program implemented in a high school biology classroom. The multimedia program incorporated concepts like teaching material through text or through audio with lean text and presenting content either with static illustrations or animations. Through these various methods, the study examined the effects of different instructional methods (text vs. audio), illustration methods (static illustration vs. animation), and spatial ability (low vs. high) on practice and posttest achievement, attitude and time. I found the results of this experiment useful for the designing and eventual pitch of Chemtris because it shows the void in the educational setting that Chemtris is attempting to fill. Because biochemistry and organic chemistry concepts are currently being taught mostly by two-dimensional textbook models or three-dimensional mechanical models, educators risk losing the interest of students and once students lose interest in learning a subject, it’s really hard to regain that interest. Having Chemtris, which focuses on three-dimensional molecular modeling that has molecules that can be rotated for better viewing as well as visual effects when molecules connect or crash onto the plane and audio effects when two molecules of opposite polarities connect, is ideal for keeping students engaged and interested the entire time.

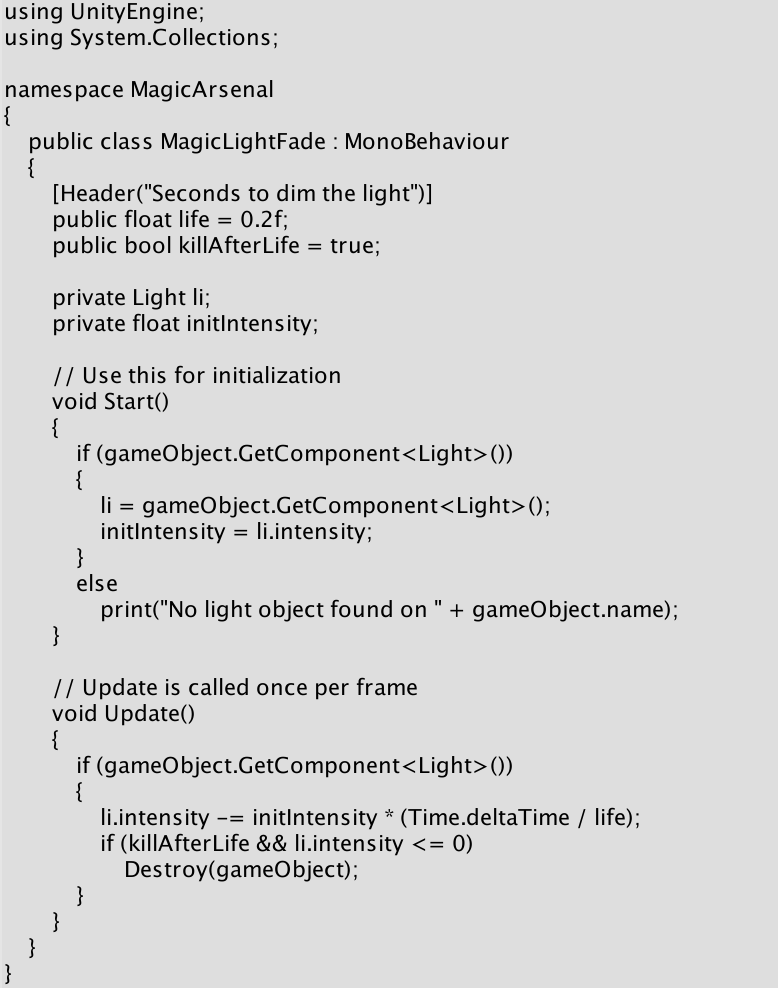
Also, collaboratively worked on implementing the audio functionality because the color modification was almost done. All I need to do with colors is see if I can add any special glow effects to the cubes so they’ll look more interesting. The code for the audio functionality is:

|  |  |
| --- | --- |
|  |  |
|  | <Project ToolsVersion="4.0" DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003"> |
|  | <PropertyGroup> |
|  | <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration> |
|  | <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform> |
|  | <ProductVersion>10.0.20506</ProductVersion> |
|  | <SchemaVersion>2.0</SchemaVersion> |
|  | <ProjectGuid>{3226CE1D-4C69-F6EB-2CE9-5EC1899E7D1B}</ProjectGuid> |
|  | <OutputType>Library</OutputType> |
|  | <AssemblyName>Assembly-CSharp-Editor</AssemblyName> |
|  | <FileAlignment>512</FileAlignment> |
|  | <ProjectTypeGuids>{E097FAD1-6243-4DAD-9C02-E9B9EFC3FFC1};{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}</ProjectTypeGuids> |
|  | <TargetFrameworkIdentifier>.NETFramework</TargetFrameworkIdentifier> |
|  | <TargetFrameworkVersion>v3.5</TargetFrameworkVersion> |
|  | <TargetFrameworkProfile>Unity Full v3.5</TargetFrameworkProfile> |
|  | <CompilerResponseFile></CompilerResponseFile> |
|  | <UnityProjectType>Editor:5</UnityProjectType> |
|  | <UnityBuildTarget>StandaloneWindows:5</UnityBuildTarget> |
|  | <UnityVersion>5.6.1f1</UnityVersion> |
|  | <RootNamespace></RootNamespace> |
|  | <LangVersion>4</LangVersion> |
|  | </PropertyGroup> |
|  | <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' "> |
|  | <DebugType>pdbonly</DebugType> |
|  | <Optimize>false</Optimize> |
|  | <OutputPath>Temp\UnityVS\_bin\Debug\</OutputPath> |
|  | <IntermediateOutputPath>Temp\UnityVS\_obj\Debug\</IntermediateOutputPath> |
|  | <ErrorReport>prompt</ErrorReport> |
|  | <WarningLevel>4</WarningLevel> |
|  |  |
|  | <AllowUnsafeBlocks>true</AllowUnsafeBlocks> |
|  | </PropertyGroup> |
|  | <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' "> |
|  | <DebugType>pdbonly</DebugType> |
|  | <Optimize>false</Optimize> |
|  | <OutputPath>Temp\UnityVS\_bin\Release\</OutputPath> |
|  | <IntermediateOutputPath>Temp\UnityVS\_obj\Release\</IntermediateOutputPath> |
|  | <ErrorReport>prompt</ErrorReport> |
|  | <WarningLevel>4</WarningLevel> |
|  |  |
|  | <AllowUnsafeBlocks>true</AllowUnsafeBlocks> |
|  | </PropertyGroup> |
|  | <ItemGroup> |
|  | <Reference Include="mscorlib" /> |
|  | <Reference Include="System" /> |
|  | <Reference Include="System.XML" /> |
|  | <Reference Include="System.Core" /> |
|  | <Reference Include="Boo.Lang" /> |
|  | <Reference Include="UnityScript.Lang" /> |
|  | <Reference Include="System.Runtime.Serialization" /> |
|  | <Reference Include="System.Xml.Linq" /> |
|  | <Reference Include="UnityEngine"> |
|  | <HintPath>Library\UnityAssemblies\UnityEngine.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEditor"> |
|  | <HintPath>Library\UnityAssemblies\UnityEditor.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEditor.Advertisements"> |
|  | <HintPath>Library\UnityAssemblies\UnityEditor.Advertisements.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEngine.UI"> |
|  | <HintPath>Library\UnityAssemblies\UnityEngine.UI.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEditor.UI"> |
|  | <HintPath>Library\UnityAssemblies\UnityEditor.UI.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEngine.Networking"> |
|  | <HintPath>Library\UnityAssemblies\UnityEngine.Networking.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEditor.Networking"> |
|  | <HintPath>Library\UnityAssemblies\UnityEditor.Networking.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEditor.TestRunner"> |
|  | <HintPath>Library\UnityAssemblies\UnityEditor.TestRunner.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEngine.TestRunner"> |
|  | <HintPath>Library\UnityAssemblies\UnityEngine.TestRunner.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="nunit.framework"> |
|  | <HintPath>Library\UnityAssemblies\nunit.framework.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEditor.TreeEditor"> |
|  | <HintPath>Library\UnityAssemblies\UnityEditor.TreeEditor.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEngine.Analytics"> |
|  | <HintPath>Library\UnityAssemblies\UnityEngine.Analytics.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEditor.Analytics"> |
|  | <HintPath>Library\UnityAssemblies\UnityEditor.Analytics.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEditor.HoloLens"> |
|  | <HintPath>Library\UnityAssemblies\UnityEditor.HoloLens.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEngine.HoloLens"> |
|  | <HintPath>Library\UnityAssemblies\UnityEngine.HoloLens.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEditor.Purchasing"> |
|  | <HintPath>Library\UnityAssemblies\UnityEditor.Purchasing.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEditor.VR"> |
|  | <HintPath>Library\UnityAssemblies\UnityEditor.VR.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEngine.VR"> |
|  | <HintPath>Library\UnityAssemblies\UnityEngine.VR.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEditor.Graphs"> |
|  | <HintPath>Library\UnityAssemblies\UnityEditor.Graphs.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEditor.Android.Extensions"> |
|  | <HintPath>Library\UnityAssemblies\UnityEditor.Android.Extensions.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEditor.iOS.Extensions"> |
|  | <HintPath>Library\UnityAssemblies\UnityEditor.iOS.Extensions.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEditor.WindowsStandalone.Extensions"> |
|  | <HintPath>Library\UnityAssemblies\UnityEditor.WindowsStandalone.Extensions.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="SyntaxTree.VisualStudio.Unity.Bridge"> |
|  | <HintPath>Library\UnityAssemblies\SyntaxTree.VisualStudio.Unity.Bridge.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEditor.iOS.Extensions.Xcode"> |
|  | <HintPath>Library\UnityAssemblies\UnityEditor.iOS.Extensions.Xcode.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="UnityEditor.iOS.Extensions.Common"> |
|  | <HintPath>Library\UnityAssemblies\UnityEditor.iOS.Extensions.Common.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="ProBuilderCore-Unity5"> |
|  | <HintPath>Assets\ProCore\ProBuilder\Classes\ProBuilderCore-Unity5.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="ProBuilderMeshOps-Unity5"> |
|  | <HintPath>Assets\ProCore\ProBuilder\Classes\ProBuilderMeshOps-Unity5.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="KDTree"> |
|  | <HintPath>Assets\ProCore\ProBuilder\Classes\ClassesLib\KDTree.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="pb\_Stl"> |
|  | <HintPath>Assets\ProCore\ProBuilder\Classes\ClassesLib\pb\_Stl.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="Triangle"> |
|  | <HintPath>Assets\ProCore\ProBuilder\Classes\ClassesLib\Triangle.dll</HintPath> |
|  | </Reference> |
|  | <Reference Include="ProBuilderEditor-Unity5"> |
|  | <HintPath>Assets\ProCore\ProBuilder\Editor\ProBuilderEditor-Unity5.dll</HintPath> |
|  | </Reference> |
|  | </ItemGroup> |
|  | <ItemGroup> |
|  | <ProjectReference Include="Chemtris.csproj"> |
|  | <Project>{436580F8-075E-CB99-D9D3-0E5DE239AE2B}</Project> |
|  | <Name>Chemtris</Name> |
|  | </ProjectReference> |
|  | </ItemGroup> |
|  | <ItemGroup> |
|  | <Compile Include="Assets\ProCore\ProBuilder\API Examples\Editor\ClearUnusedAttributes.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\API Examples\Editor\CreateShadowObject.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\API Examples\Editor\CustomAction.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\API Examples\Editor\EditorCallbackViewer.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\API Examples\Editor\RenameNewObjects.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\API Examples\Editor\SetCustomUnwrapParams.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\About\Editor\pb\_AboutWindow.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\About\Editor\pb\_UpdateAvailable.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\About\Editor\pb\_UpdateCheck.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\About\Editor\pb\_VersionInfo.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\About\Editor\pb\_VersionUtil.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Actions\pb\_GenerateUV2.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Geometry\pb\_DetachDeleteFace.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Geometry\pb\_ExtrudeElement.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Geometry\pb\_FlipEdge.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Geometry\pb\_FlipNormals.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Geometry\pb\_SetPivot.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Repair\pb\_CleanLeakedMeshes.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Repair\pb\_ForceSceneRefresh.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Repair\pb\_RebuildMeshes.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Repair\pb\_RebuildSharedIndices.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Repair\pb\_RemoveDegenerateTris.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Repair\pb\_RepairColors.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Repair\pb\_RepairMeshReferences.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Repair\pb\_RepairUV.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Repair\pb\_ResetEntityMaterials.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Repair\pb\_UpgradeBasicToAdvanced.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Selection\pb\_EdgeSelection.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Selection\pb\_ExpandSelection.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Selection\pb\_InvertSelection.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\MenuItems\Selection\pb\_MaterialSelection.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\pb\_MenuItems.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\pb\_MissingScriptEditor.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\pb\_Preferences.cs" /> |
|  | <Compile Include="Assets\ProCore\ProBuilder\Editor\pb\_ScenePostProcessor.cs" /> |
|  | </ItemGroup> |
|  | <ItemGroup> |
|  | <None Include="Assets\Mirza Beig\README - Inferno VFX.txt" /> |
|  | <None Include="Assets\ProCore\ProBuilder\API Examples\Icosphere FFT\README.txt" /> |
|  | <None Include="Assets\ProCore\ProBuilder\About\Credits.txt" /> |
|  | <None Include="Assets\ProCore\ProBuilder\About\Font\SIL Open Font License.txt" /> |
|  | <None Include="Assets\ProCore\ProBuilder\About\changelog.txt" /> |
|  | <None Include="Assets\ProCore\ProBuilder\About\pc\_AboutEntry\_ProBuilder.txt" /> |
|  | <None Include="Assets\ProCore\ProBuilder\Shader\DiffuseTextureBlend.shader" /> |
|  | <None Include="Assets\ProCore\ProBuilder\Shader\DiffuseVertexColor.shader" /> |
|  | <None Include="Assets\ProCore\ProBuilder\Shader\UnlitSolidColor.shader" /> |
|  | <None Include="Assets\ProCore\ProBuilder\Shader\pb\_FaceHighlight.shader" /> |
|  | <None Include="Assets\ProCore\ProBuilder\Shader\pb\_FacePicker.shader" /> |
|  | <None Include="Assets\ProCore\ProBuilder\Shader\pb\_HideVertices.shader" /> |
|  | <None Include="Assets\ProCore\ProBuilder\Shader\pb\_ReferenceUnlit.shader" /> |
|  | <None Include="Assets\ProCore\ProBuilder\Shader\pb\_ScrollHighlight.shader" /> |
|  | <None Include="Assets\ProCore\ProBuilder\Shader\pb\_SelectionPicker.shader" /> |
|  | <None Include="Assets\ProCore\ProBuilder\Shader\pb\_StandardVertexColor.shader" /> |
|  | <None Include="Assets\ProCore\ProBuilder\Shader\pb\_TransparentOverlay.shader" /> |
|  | <None Include="Assets\ProCore\ProBuilder\Shader\pb\_UnlitVertexColor.shader" /> |
|  | <None Include="Assets\ProCore\ProBuilder\Shader\pb\_VertexPicker.shader" /> |
|  | <None Include="Assets\ProCore\ProBuilder\Shader\pb\_VertexShader.shader" /> |
|  | <None Include="Assets\ProCore\ProBuilder\readme.txt" /> |
|  | </ItemGroup> |
|  | <Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" /> |
|  | <Target Name="GenerateTargetFrameworkMonikerAttribute" /> |
|  | </Project> |

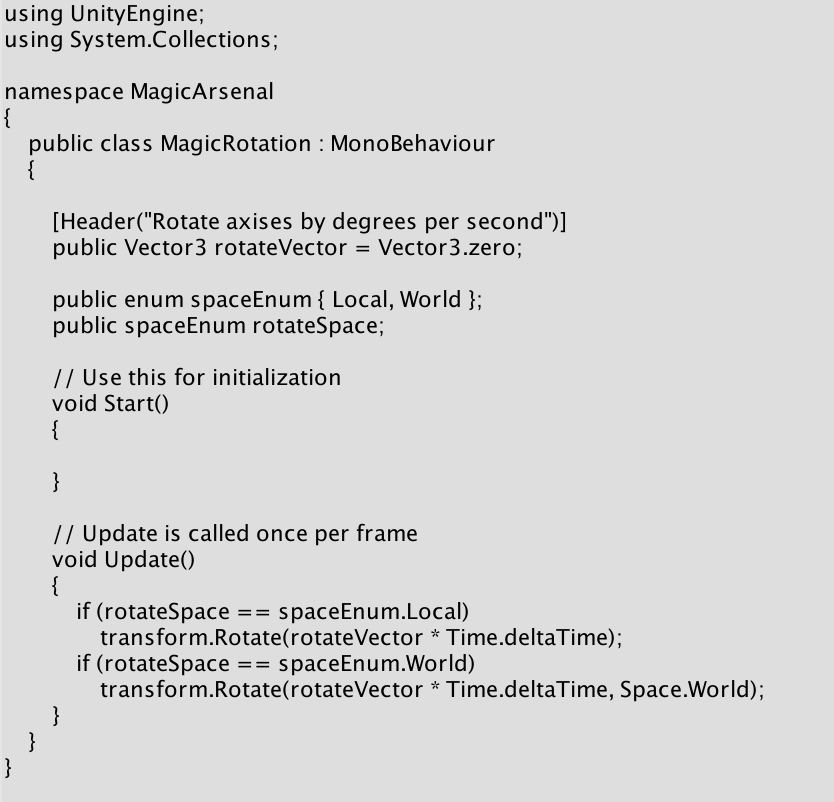
*Day 3: July 12th:*

Spent a majority of the day continuing to find relevant articles and beginning to work on the poster which is due next week. I found an article by Judy Strauss and Donna J. Hill called Student Use and Perceptions of Web-Based Instructional Tools: Laggards in Traditional Classrooms. In this article, they demonstrate how many students in traditional classrooms do not make use of web-based instruction (WBI) tools for better and faster learning. The results of the study show that the students who are “heavy users” or the students who take advantage of the web-based instructional tools available in the classroom for usage and better understanding of the subject material, expressed a higher satisfaction with classes who are implementing WBI tools and rated their competency and perceived learning higher than “light users” or the students who rely strictly on the traditional teaching methods of the teacher. The students who used tools found on the web such as e-mailing the teacher any questions they may have about the subject material, reading excerpts from the online textbook, or taking tests online benefitted a lot more and received a significantly more well-rounded and useful learning experience than those who did not use online tools. This article was useful because it shows that Chemtris, when implemented in classrooms, can actually fill a significant gap in learning and can be successful.

I continued to work on fixing the audio and adding sound effects. I also worked on code for the magic effects that can be used in Unity such as light fade, light flicker, and rotations. The code for the light fade is:



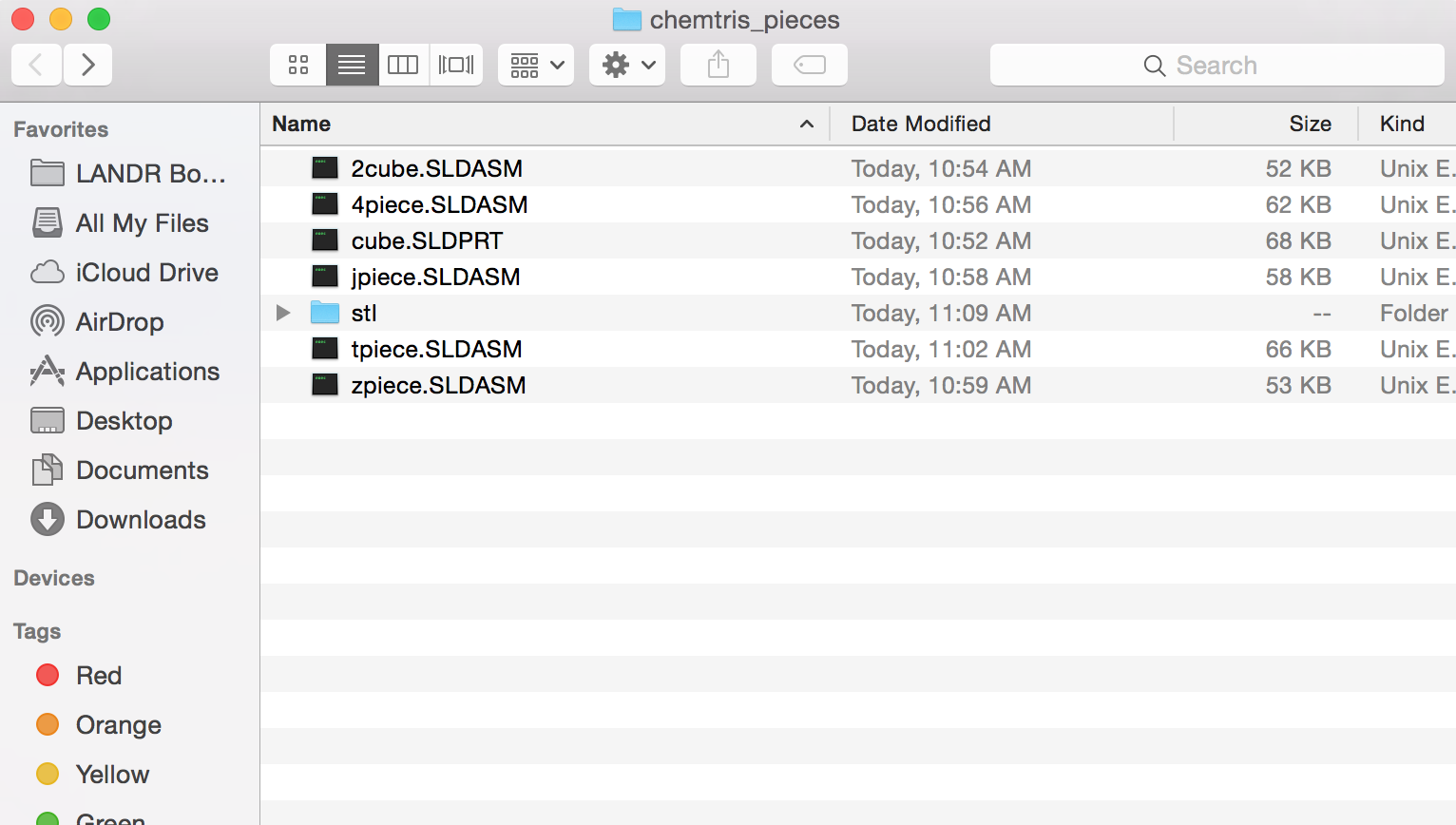
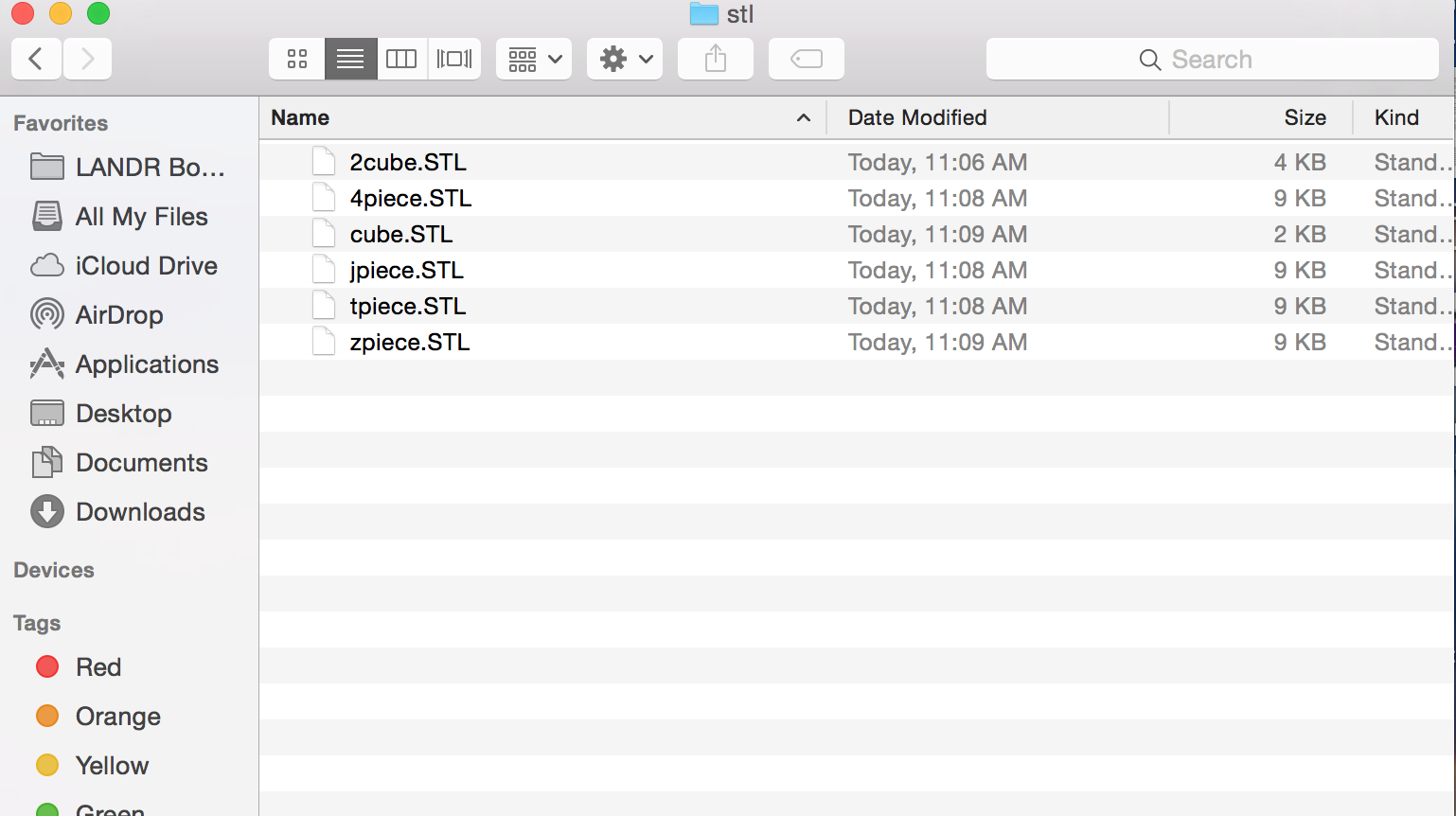
The code for the rotation is:



*Day 4: July 13th:*

I worked on adding different tracks of audio to different scenes of the game especially the game menu and different levels as well as sound effects for when the pieces hit the plane or when the pieces crash into each other. I found a real nice arcade music background that I added to the game and committed the new version to github.

Eugene and I went to the MakerSpace and tried to work on finding a 3D printing software that we can work with in conjunction with the Ultimaker’s Cura software to print different tetrominoes for the poster presentation. We created the design for all of the pieces but we still need to see if the MakerSpace has the colors we need and if we can reserve a printer because the printing might take a while. We made sure both of us had the STL files for all the pieces so either one of us could print the pieces when we have the chance.



*Day 5: July 14th:*

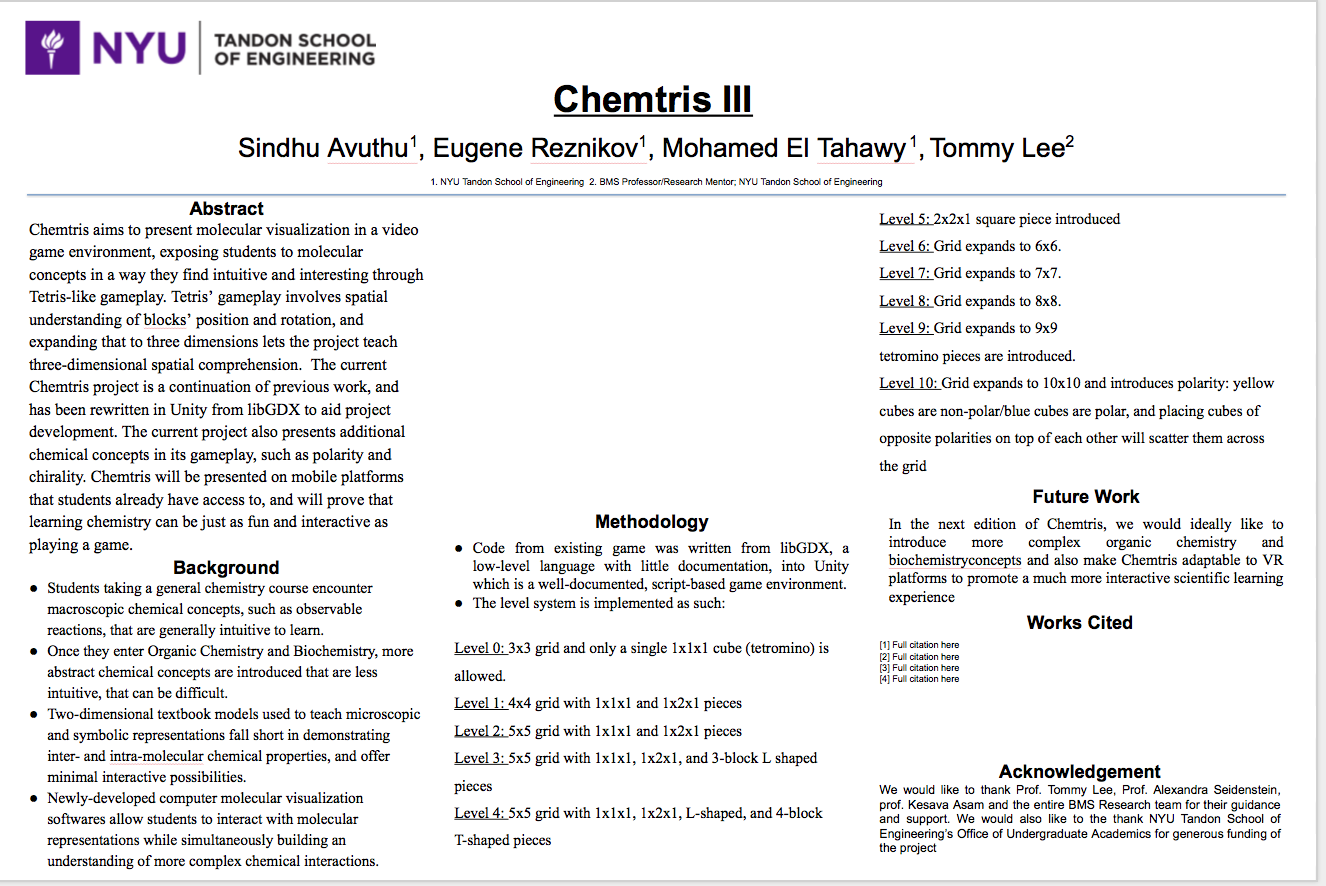
I worked on researching possible undergraduate journals that this work could be published in or even patents that we could get for this work. I found AJUR (American Journal of Undergraduate Research). American Journal of Undergraduate Research (*AJUR*) is a national, independent, peer-reviewed, open-source, quarterly, multidisciplinary student research journal

The following link provides the guidelines for Submissions and Review for this journal:

<http://www.ajuronline.org/submissions/>. According to this, publishing an article into this journal is a very long process which includes writing a scientific paper with an abstract, background, method, results, discussion, conclusion and any relevant pictures or graphs illustrating the results. It also includes attaching a list of seven potential reviewers for the manuscript who should be faculty and experts in the field of the manuscript. They may not be our former or current collaborators/ associates/ co-authors/ teachers/ mentors/ mentees. It also requires the faculty research mentor to review and edit the manuscript and send a separate email to the journal from an institutional account giving permission for our work to be reviewed and published by AJUR. The editor will not respond until faculty sponsor’s/mentor’s note is received. If sponsor’s/mentor’s support is not provided within ten days, the submission is automatically rejected.

I also found the Journal of Undergraduate Research and Scholarly Excellence (JUR) which is a peer-reviewed, undergraduate journal registered with the Library of Congress that accepts submissions of any subject, from any undergraduate institution. More information about submission and review guidelines could be found here: <http://jurtest.colostate.edu/submit/>. Even though all of these publishing endeavors are long processes that could take up to 12 to 16 weeks, publishing is definitely a good way to share this research with anyone who’s interested and it is also a great thing to have on your resume so Eugene and I , with the help of Mohamed and Prof. Lee will hopefully try to get this work published.

The remaining of the day I worked on the poster that we need to send to BMS Coordinators by Monday so we can send it to SURP and Sara-Lee by Friday. So far, I worked on putting in all the information and changing it t buller-form so it’s easier to follow and more eye-catching. Eugene and I will work together to add pictures from the current version of the game. For the presentation on August 4th, we are hoping to have a playable version on an Iphone and and Android for display and also the 3D printed version of the tetrominoes on display.



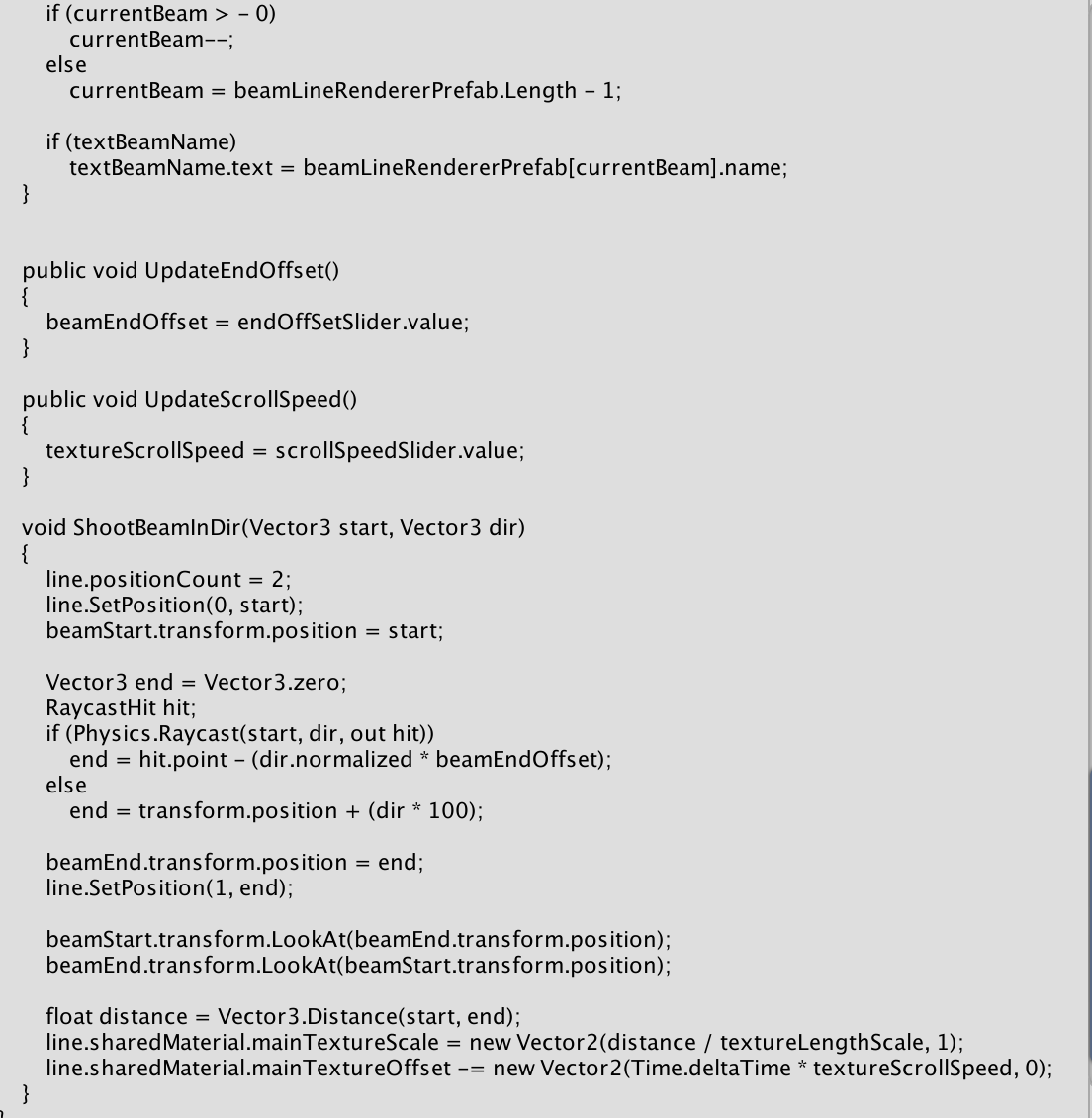
**Week 7: Week of July 17th**

*Day 1: July 17th:*

Worked on the code for magic beam script:





**

Also modified the code for the light flicker mechanic:

*// Properties*

var waveFunction : String = "sin"; *// possible values: sin, tri(angle), sqr(square), saw(tooth), inv(verted sawtooth), noise (random)*

var base : float = 0.0; *// start*

var amplitude : float = 1.0; *// amplitude of the wave*

var phase : float = 0.0; *// start point inside on wave cycle*

var frequency : float = 0.5; *// cycle frequency per second*

*// Keep a copy of the original color*

private var originalColor : Color;

*// Store the original color*

function Start () {

originalColor = GetComponent(Light).color;

}

function Update () {

var light : Light = GetComponent(Light);

light.color = originalColor \* (EvalWave());

}

function EvalWave () {

var x : float = (Time.time + phase)\*frequency;

var y : float;

x = x - Mathf.Floor(x); *// normalized value (0..1)*

if (waveFunction=="sin") {

y = Mathf.Sin(x\*2\*Mathf.PI);

}

else if (waveFunction=="tri") {

if (x < 0.5)

y = 4.0 \* x - 1.0;

else

y = -4.0 \* x + 3.0;

}

else if (waveFunction=="sqr") {

if (x < 0.5)

y = 1.0;

else

y = -1.0;

}

else if (waveFunction=="saw") {

y = x;

}

else if (waveFunction=="inv") {

y = 1.0 - x;

}

else if (waveFunction=="noise") {

y = 1 - (Random.value\*2);

}

else {

y = 1.0;

}

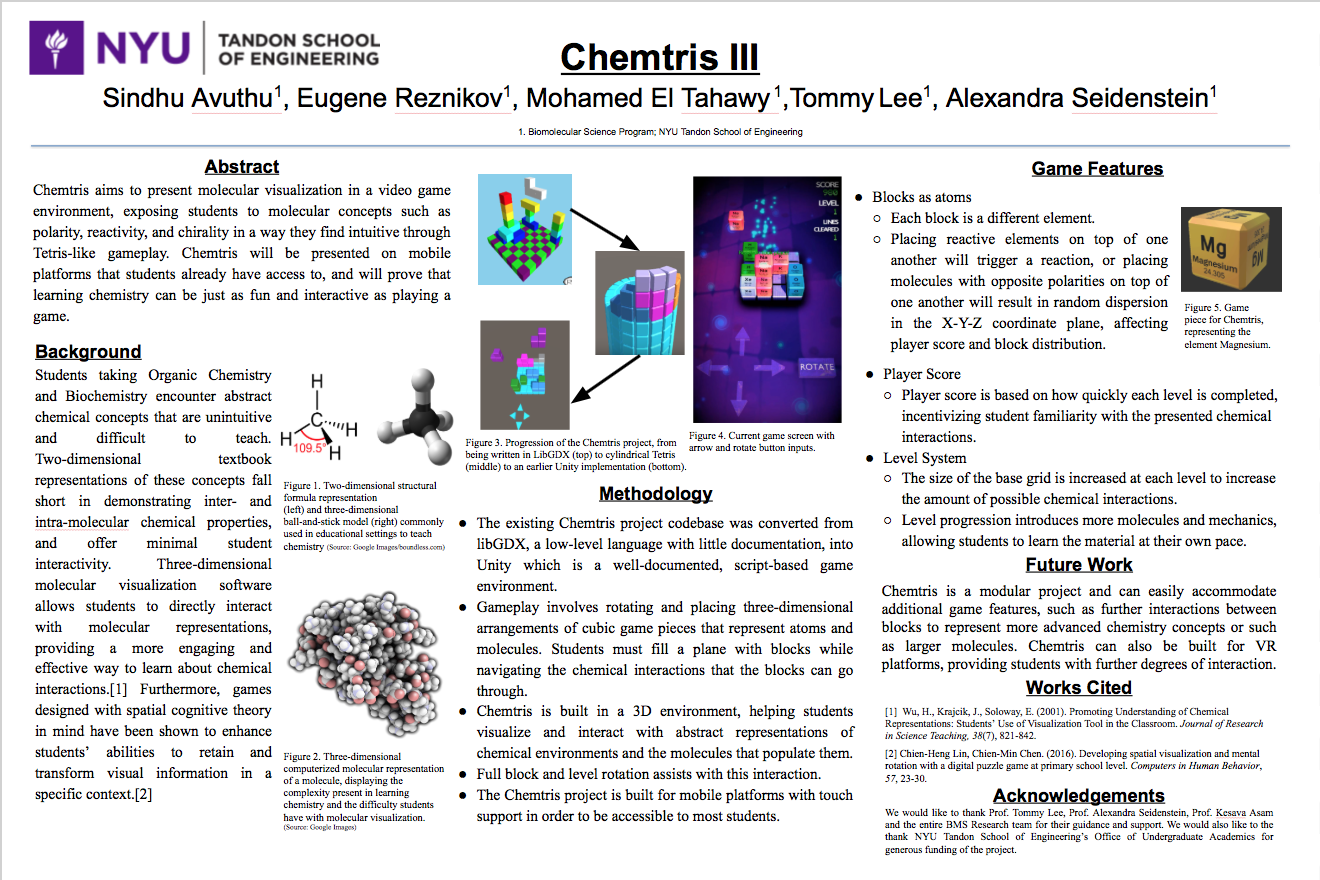
return (y\*amplitude)+base;

}

*Day 2: July 18th:*

I worked on fixing the poster template for a majority of the day. We had a lot of comments about reducing the amount of words and increasing the number of pictures on the poster to make it more attracting to the audience. We reduced the abstract section from the entire 200+ words we submitted for the research abstract book to 2-3 sentences. We also compressed the background section and added two pictures, one comparing the two-dimensional structural formula of molecules often found in textbooks and the three dimensional ball-and-stick model of molecules often found as a supplementary tool in classrooms, and one of the three-dimensional computerized molecular model which shows how difficult it is for students to learn abstract chemistry concepts.

We also added three pictures showing the progression of the game from its first form (by MD Kabir) to its spherical form by (Mohamed A-El Tahway) to it’s previous version in Unity. We placed these pictures next to a screenshot of the current game in action. We also added a picture of the elements that are available in the game such as elements of Magnesium, Hydrogen, Oxygen and noble non-reactive gases such as Argon and Neon.



I also worked on modifying the 3D versions of the molecules that we still need to print out. We might eventually need help from one of the 3D printing groups but we’re not sure how long this will take to fix and print but it shouldn't take long because they are only cubes.

*Day 3: July 19th:*

I worked on the article reviews for this week today. I found an article called Developing and Using Conceptual Computer Animations for Chemistry Instruction by K.A. Burke, Thomas J. Greenbowe, and Mark A. Windschitl in which they discuss the positive impact of having multimedia technology in the classroom on the teaching efficiency of educators and the ability of students to learn as best as they can. Multimedia technology has been very abundant in the educational spectre in the past couple of years because it allows chemistry instructors to teach in their own way through computer animations making it easier for them to teach but it also makes it easier for students to learn the material because the complex concepts are presented in a computer format. This makes the molecular representations extremely accurate, down to the intermolecular, intermolecular and dimensional properties of molecules that are often hard to understand through two-dimensional textbook models or three-dimensional ball-and-stick models. Allowing instructors to make their own conceptual computer animations allow them to help students understand basic concepts or principles regarding dynamic chemical processes.

Throughout their paper, Burke, Greenbowe and Windschitl discuss several issues revolving the development and use of conceptual computer animations that either allow or prevent the abundant use of them in educational settings. They discuss the possibility of linking a conceptual computer animations sequence to a lecture demonstration of new chemistry material, allowing students to be exposed to microscopic, macroscopic and symbolic levels of representation. Because computer animations give instructors the opportunity to present topics through the particulate nature of matter view promoting conceptual understanding, the authors of this article suggest combining a design team filled with experts of content area, instructional design, computer animation techniques, and graphic and sound production, with chemistry instructors to plan out, develop, and produce the animation sequences. Due to the abundant nature of existing drawing, animating, and audio-animation combination software tools, producing these computer chemical animations shouldn’t be a difficult task. They also discuss ways to allow students to have a 24/7 access to these instructional computer animations through the internet either via the world wide web or to local file servers through intranets. Instructors and developers should also be aware of the copyright laws pertaining to the educational use of images, illustrations and sounds found on the internet. Finally, the authors stress that incorporating computer animations into the lesson while omitting certain topics in the curriculum is consistent with reports from the ACS task force that emphasizes "less is more" in terms of curriculum reform.

I also found another article called Adapting to Student Learning Styles: Engaging Students with Cell Phone Technology in Organic Chemistry Instruction by David P. Pursell in which he explains how important it is for instructors to be dynamic and be willing to constantly adapt to different student learning styles and mediums. Traditionally, organic chemistry students make 3x5 inch flashcards to aid in their learning of nomenclature, structures and reactions. Because flash cards are becoming abundant on computers, the opportunities for constant drilling and practice or organic chemistry concepts are endless. However, students of the current generation are spending less time on computers but are almost always on their cell phones because they’re always with them. Throughout this article, Pursell suggest an even more recent educational technology initiative that is in line with this transition from computers to cell phone, which involves using cell phone flash cards to help students learn organic chemistry nomenclature, structures, and reactions. He mentions that student attitudes toward using cell phone flashcards to learn difficult concepts were positive.

This article suggest the use of flashcards available on mobile platforms to promote learning difficult organic chemistry concepts. No matter how effective these flash cards might be, students need to be interested enough to pick these flashcards to learn from over social media or any of the other games available on their phones. Chemtris allows students to learn all these organic chemistry and biochemistry concepts, especially reactivity, polarity, and chirality, in a fun, intuitive, and engaging way. With bright colors, cool mechanics and graphics, and upbeat background music, Chemtris makes students want to learn chemistry, making it different from all the other educational programs out there currently.

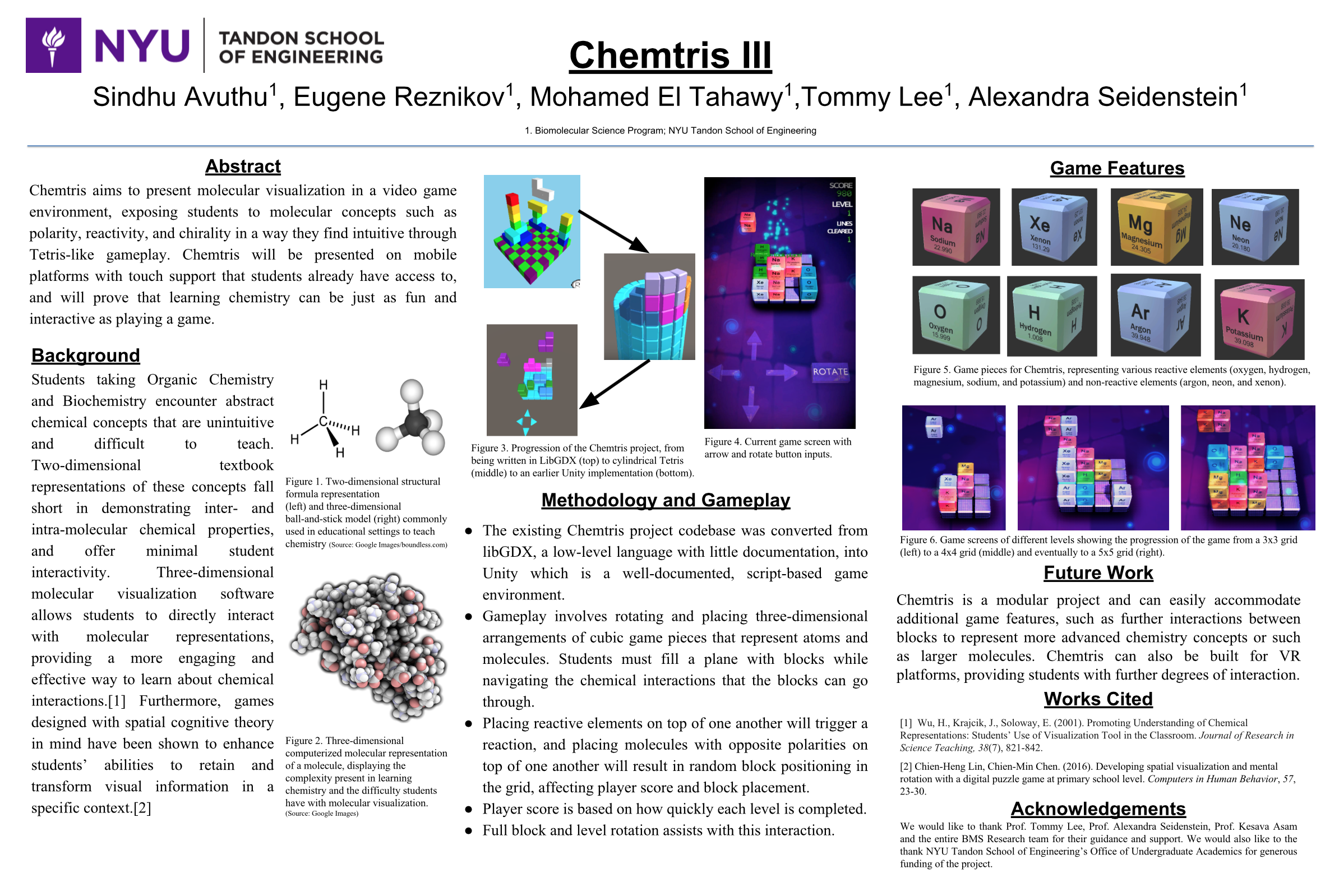
I also began to work on the WIP #2 powerpoint with Eugene and fixed up the poster a little bit.

*Day 4: July 20th:*

I spent most of today working on my poster and my work in progress with Eugene. Eugene and I also spend two hours in the MakerSpace trying to print the tetrominoes but we ran into a lot of difficulties because 1) almost all of the printers were being used for long-term projects 2) we found one printer that was available and placed the color that we wanted in the back but the printer wasnt detecting the SD card that had our files on it and not other SD cards were available (turned out that the SD card was broken) 3) we had to download Cura onto our laptops and try to print using Raspberry Pi but that didnt work either. Eugene and I are going to try again next week to print out the pieces.

The WIP#2 was mostly the same as the first one with changes made to the progress section, the pictures, and the to-do list.

We also updated our poster after the WIP and we added more pictures to make the poster as colorful and attractive as possible. This is our current poster:



*Day 5: July 21st:*

Eugene and I again tried to go to the Makerspace and we spent a majority of the day trying to print the tetrominoes but we encountered many problems again like an insufficient number of available 3D printers due to the fact that many people have very long printing projects that take up upwards of 5 hours. We asked one of the people working in the Makerspace to help us and he helped us in the beginning by helping us setup the printer with the proper colors (we wanted to print the cubes in yellow) but there were still no SD cards available for use so we will come back again sometime next week to work on them and hopefully successfully print them before August 4th.

**Week 8: Week of July 24th**

*Day 1: July 24th:*

I spent a majority of the day today working on fixing up the code for some of the features of the game. The code currently is kind of messy because there are many parts where we can make the code more efficient and make the game run faster with no lag-time if we take out places that repeat code and instead replace it with functions so I worked on that today.

I also fixed the audio because the background music wasn’t loading properly and you couldn't hear it well along with the special sound effects that are accompanied with certain game mechanics so I also modified that a bit.

*Day 2: July 25th:*

In the morning, we had the regular check-in with Prof. Seidenstein and Mackensie where we updated her with our projects. We then went to her talk about choosing a path that you’re passionate about. I really loved Prof. Seidenstein’s talk and thought it was definitely the best one we’ve had so far because it was relatable, funny, and useful. I feel like I learned a lot about her and also learned that the path to success isn’t always clear-cut and straight, even though it would be real nice if it was. I was shocked that she volunteered at the Mother Teresa home in Calcutta because that has been something I’ve wanted to do since I was very young and would like to do so sometime soon. I’ve always looked up to Prof. Seidenstein and saw her as someone who has everything figured out and is super successful, which she is but after her presentation, I saw that she spent a lot of time trying new things and venturing into new career fields, volunteering, travelling, and it just made me look up to her even more. One piece of advice of hers from the presentation that really stuck with me is when she said “no matter what you do or where you go, you’ll always end up doing what you really love to do.”

*Day 3: July 26th:*

I spent today working on finding articles for this week’s article reviews. I found an article written by Zahira Merchant, Ernest T. Goetz, Wendy Keeney-Kennicutt, Oi-man Kwok, Lauren Cifuentes, and Trina J. Davis that discuss the problems associated with learning science in a high school or college setting in their article called The learner characteristics, features of desktop 3D virtual reality environments, and college chemistry instruction: A structural equation modeling analysis. In the study represented in this article, they examine the positive and negative impacts of a 3D desktop virtual reality environment on the learner characteristics. Learner characteristics consist of perceptual and psychological variables that can significantly improve the chemistry-related learning achievements of students in an introductory college chemistry class. The study shows that there is a direct positive correlation between the usage of 3D virtual reality technology in classrooms and the scores on the chemistry learning test because the 3D virtual reality technology in an educational setting effectively combines selected perceptual variables such as spatial orientation and usability as well as psychological variables such as self-efficacy and presence. These variables were tested and analyzed in the study using the structural equation modeling approach and the results of the test showed that the usability strongly moderated the interaction between three-dimensional virtual reality features, self-efficacy, presence, and spatial orientation. Out of all the features, spatial orientation and self-efficacy were the two that had significantly higher statistics and also a stronger correlation to positive impact on the chemistry learning test. The results of this study showed that 3D virtual-reality based instruction is ideal because it enhances students’ chemistry achievement. The study also shows that the research model derived from this experiment could be contributed to classrooms to enhance the the effectiveness of desktop virtual reality environments in an educational setting. This study was also useful in proving to science educators and instructional and multimedia designers and developers that instructional design principles could be used in science-based instruction that is effective and interesting.

*Day 4: July 27th:*

After the check-in, Eugene and I went to the Makerspace to work on printing the tetrominoes. After some back and forth and switching between printers and SD cards because almost all of them were in use, we finally found a printer and started the print job. It said it would take 7 hours to print a T-cube so we decided to check up on it tomorrow or pick it up from the front next week.

I spent the rest of the day working on the article reviews. I found an article by Dilek Ardac and Ali Hasan Sezen called Effectiveness of Computer-Based Chemistry Instruction in Enhancing the Learning of Content and Variable Control Under Guided Versus Unguided Conditions in which they analyze the impacts of computer technology in teaching and learning chemistry educational content. They discuss the benefits of a computerized learning environment and how using technology in the classroom can offer several possibilities that can be used to improve the teaching of content along with the process. Their research in this article shows that additional guidance in the form of computer-based instruction is beneficial to students because it encourages them to actively construct their understanding of the information presented to them. Throughout this study, Dilek and Ali also examine guided and unguided computer based instruction and examine how effective they are with respect to regular traditional classroom instruction. They studied the effectiveness of these teaching techniques by analyzing changes in improvement of content knowledge and process skills among students with low and high chemistry achievement levels. The results of this study showed that effective computer-based instruction is highly beneficial to students when it is paired with teacher-directed guidance. They also discovered that computer-based instruction, regardless of whether it was with or without guidance, was observed to be more effective than regular instruction in terms of teaching students the process skills that are vital for high chemistry achievement. Finally, they discovered that students with unguided instruction did not construct the expected content knowledge as well as student who received regular or guided computer-based science instruction.

*Day 5: July 28th:*

Eugene and I went to pick up the T-cube from the makerspace and modified it a bit and also modified the design for the other tetrominoes that we’re planning to print later. When we printed the T-cube, we used whatever printer that was available because almost all of the printers were in use already and we didn't really check the color on this printer. The T-cube is supposed to be bright yellow but it ended up printing black. We’re still deciding between leaving the cube the way it is or painting it the color we need it to be.

**Week 9: Week of July 31st**

*Day 1: July 31st:*

I worked on researching more articles to help us modify the game and make it better. I also worked on the game and fixed the problems we had with getting it to not crash on mobile platforms because we’re trying to have it available on an Android phone during the presentation on Friday.

*Day 2: August 1st:*

Eugene and I went to the Makerspace and began the print job for the Z shaped tetromino with the only color that was available in the printer which was white/gray. It said the print job is going to take 11 hours so we’re going to pick it up tomorrow and start printing another piece hopefully so that we have 3D printed tetrominoes to show for the poster presentation

*Day 3: August 2nd:*

We went to pick up the Z-shaped tetromino and it turned it okay. I spent the day mostly researching articles to see if we can do anything else to fix the project before Friday. I found an article called Using Three-Dimensional Models to Teach Molecular Structures in High School Chemistry by Cynthia F. Copolo and Paul B. Hounshell in which they analyze the effectiveness of two-dimensional molecular modeling and three-dimensional molecular modeling in teaching high school students concepts in chemistry. A molecular model is a physical model that accurately represents molecules and molecular processes in inorganic chemistry, organic chemistry and cellular biology. Molecular modeling and the computer version of molecular modeling, called molecular graphing, uses mathematical models, graphical depictions, and techniques to simplify the electronic structure of molecules. All throughout primary education, two-dimensional molecular models have been prominent in teaching children molecular interactions and basic concepts of chemistry and biology. However, using three-dimensional models has proven to be much more effective due to the visualization of the actual structure and the ability to move the model around 360 degrees and see molecular interactions in detail.

Throughout the study, they compare the effects of two-dimensional textbook representations, three-dimensional computer models, three-dimensional ball and stick models, and combination of ball and stick models with computer molecular models on the students’ results on the isomeric identification and isomeric retention tests as well as their overall understanding of chemistry concepts. The study also tested whether students who learned chemistry through three-dimensional modeling could translate what they learned into understanding two-dimensional models. Organic chemistry is often taught by using two-dimensional organic molecules using the planar of the Fisher projection technique. Often times, geometric shapes of molecules are depicted in a way in which the central atom is surrounded by single/double/triple bonds and the bonds are generally vertically aligned and horizontal directed toward the observer. Planar projections perfectly depict functional group distribution but other dimensional qualities are not properly shown through these models. Hand-held mechanical models are ideal but they are also limited in scope and structural information can be distorted by the person moving around the model manually. Mechanical models also do not do justice to bond energies and their dynamic properties. Computer generated models of molecules depict molecules in three dimensions and also give the user the ability to twist, rotate animate, and invert the molecules while staying true to the bond energies that are present in different conformations of the molecules. Allowing children to manipulate the molecule without allowing them to alter the chemical properties and accuracy of the molecules is an effective way to teach organic structures and isomers through abstract concepts. It allows students to successfully conceptualize abstract properties of the molecule while allowing students to view the model through different angles.

*Day 4: August 3rd:*

I continued researching more articles. I found an article by Nitza Barnea and Yehudit J. Dori called Computerized Molecular Modeling as a Tool To Improve Chemistry Teaching in which they test the effectiveness of computerized molecular models (CMM) in an educational setting by setting up a 14 hour workshop in which CMM was incorporated into a pre and in-service educational training in the Department of Education in Technology and Science at the Technion. Most teachers had a favorable reaction to CMM but were hesitant to actually use it in the classroom due to an anxiety to work with computers/insufficient access to technology. They also tested CMM in three experimental 10th grade classes with two control classes and the results showed that the experimental group scored higher on tests that tested the students’ understanding of concepts related to chemical structure and bonding as well as geometric and symbolic representation.

Over the years, the methods used for teaching subject material in classrooms has changed dramatically. What once used to be an educational setting that was dominated by two-dimensional models, educators are beginning to use computerized molecular modeling in classrooms as a way to teach chemistry because of the convenience and simplicity that is associated with this new platform. Computerized molecular modeling or CMM makes it easy for students to construct molecules in any size, color or quantity and it also makes it easier for students to visualize matter in terms of a particulate model.

Three-dimensional hand-held molecular models have been often used in classrooms to depict molecules however these models are only a simulation and bears no resemblance to an actual molecule. This can result in students only having a partial or incorrect understanding of chemistry concepts. Also, using such models only adds to the existing difficulties students have when it comes to making linkages between macroscopic, submicroscopic and symbolic representations. Therefore, in order to solve the problem students have with making referential linkages between various representations and different dimensions (two-dimensional and three-dimensional), it would be ideal to introduce computer graphics or computer generated molecular models to the classroom along with the existing educational tools such as models and analogies to further enhance the students’ learning experiences. Classrooms as well as the drug industry are using computer programs to accurately represent molecular structures and calculate their properties in organic chemistry and biochemistry. The benefits of using computerized molecular modeling gives students the flexibility to create any molecules of any size, number, and model type and it also allows models to be constructed with different color patterns and precise size to improve the accuracy of the molecules.

*Day 5: August 4th:*

Today was the day of the poster session. We did some last minute practicing, fixing up and printing pieces before the poster session. Overall, I think the poster presentation went well. A good amount of people came by our poster because they saw the colorful pictures and wanted to learn more. Eugene and I took turns pitching the project and we got some good feedback to make the game even better. It was also easy to explain the game to people who have no background in coding or chemistry education because the game itself is relatively straightforward. Also having the game on a laptop available to play/demonstrate made it much more interesting.

**Week 9: Week of August 7th**

*Day 1: August 7th:*

We decided that this week is going to be a lot of wrapping up things and doing any last minute touch ups to the game mechanics before we hand it off to the next team. We started gathering all the things we wanted to put into the folder like all of the past WIPs, abstract, lab notebooks, article reviews (two from every week), poster pdf, stl files for the 3D tetrominoes, code for the game mechanics, send off letter etc.

*Day 2: August 8th:*

I worked on the final WIP