USF Independent Study

Name: Kevin Wagner

UID: U7572-3519

Email: Wagnerk1@mail.usf.edu

Teacher: Yicheng Tu

Software Project

General Game idea and inspiration

Starting this project, the initial inspiration was to a create a Galaga-style shooter that would teach basic math to children. Below is a screenshot of Galaga to get an idea what the genesis of the inspiration is for those not familiar with Galaga.

Galaga Screenshot. Digital image. *Gamefabrique*. N.p., n.d. Web. 1 Nov. 2015. <gamefabrique.com>.

The general idea is a player ship facing waves of incoming enemies destroying them to get the highest score possible before it is inevitably destroyed. The goal of this project is to take this simple game idea and inject an educational element. By making a Galaga-style game where destroying enemy ships requires the player to solve simple math problems in addition to accurately targeting their enemy. An educational game that is similar to what this project is trying to accomplish is a section of the game Math Blaster, where the player needs to shoot space debris and various trash

that has a number that is the solution to a particular math problem. This idea of shooting the enemy or object that matches the correct answer to an equation presented to the player is the method my project uses. The difference is, it applies it to a Galaga style combat system, whereas in Math Blaster the player is in a first person view, and shooting harmless garbage in space facing no retaliation.

Screen shot of the Sega Version of Math Blaster. Digital image. *Consoleclassix*. N.p., n.d. Web. 1 Nov. 2015. <http://www.consoleclassix.com/info_img/Math_Blaster_Episode_One_GEN_ScreenShot2.jpg>.

With the general idea and inspiration for the project locked down, the only thing left was to give it a working title and proceed to figure out the details of the design and scope. The current working title assigned to this project is Laser Defender and the project will from now on out be referred to as Laser Defender.

Laser Defender Design considerations and scope

The design of Laser Defender went through several iterations, which have been condensed into two main versions of the game over its development. Below I will discuss how I designed the game and what changes I made over the two prototype milestones. Before I could even consider designing, I needed to layout a scope for the project. Without a realistic scope, software projects

have a tendency never to end, so to avoid this a realistic scope for the prototype of Laser Defender is necessary. The Laser Defender prototype will be limited to four scenes or screens. These include a main menu where the player can start the game or go to the option that allows the player to tweak the game settings. A play screen where the player fight enemies, until they are destroyed, and a game over screen where the player can see their score and chose to go back to the main menu screen.

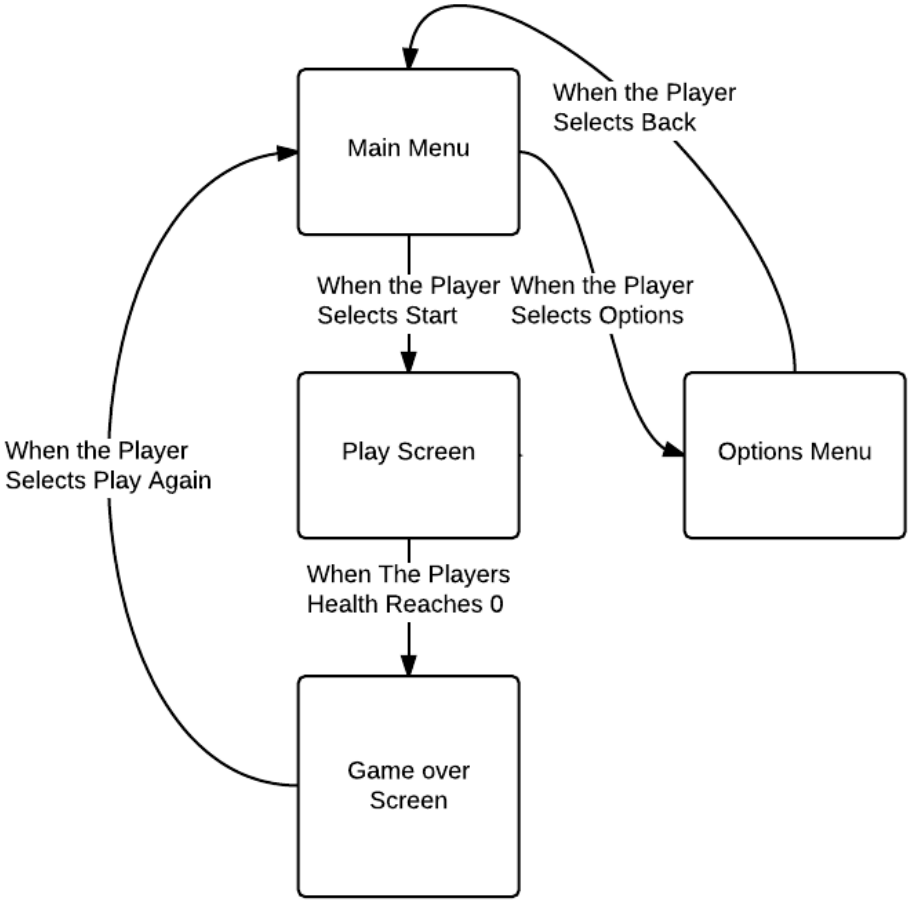


Diagram showing the screen flow.

After overall structure and scope of the project was decided, on I began creating the game. Below we shall see the first version of the prototype.

Laser Defender 1st Version

In the first version of Laser Defender, only three screens are available: Main Menu, Play, and Game Over screen.

1. Main Menu Screen

Laser Defender Main Menu V1

The initial design of the main menu screen is simple; in this version of the game, it has only one option, to select start and continue to the play screen. The only other function it performs is to start the background music that will continue to play through all the screens on loop. When you select start, you are taken to the play screen that is more complex.

2. Play Screen

Laser Defender Play Screen V1

The above screen is what the player is presented with when they hit the start button on the main menu. First, the player's ship is now visible and controllable, the player can move anywhere within the bounds of the screen with the arrow keys and press the space bar to shoot a projectile. Second, we have enemies that fly in from the top of the screen, firing back at random intervals and moving from side to side to avoid the player's return fire. There are also several UI elements: a score counter that increases by 100 for every enemy ship destroyed, a health counter that decreases by 100 for each hit sustained, and a laser power element that shows the current equation, the answer to which is a ship you can destroy. If you hit a ship that is not the answer to the equation in Laser Power, the shot will not destroy them. Every time you destroy the correct ship, the equation will change so that the answer is now a different ship. This process will continue until all the ships are destroyed, after which a new wave will descend. This process will repeat until the player reaches

zero health and is moved to the game over screen. The other element of the screen is the scrolling starfield in the background. This starfield is constructed from two different particle systems. A particle system that creates the large bright particles and another that creates the smaller and dimmer particles. The bright particles move faster than the smaller particles making them seem closer and adding a sense of depth to the background.

3. Game Over Screen

Laser Defender Game Over Screen V1

4. Interesting bits of code

A. The Equation Generation Script

At this point in the project, the equation generation code is very basic. It works by taking in an answer, which is the number from a randomly selected ship and then through integer division generates a simple math equation that is then placed in the Laser Power UI element.

```
// Generate a math problem
public void GenerateMathProblem(int awnser)
{
    //divide method creates a two variable addition problem
    int Var1 = awnser / 2;
    int Var2 = awnser - Var1;
    LaserPowerText.GetComponent<Text>().text = "Laser Power: " + Var1.ToString() + " + " + Var2.ToString();
}
```

Laser Defender Equation Generation Function

B. Player Controller Script

The player control has many functions, two of the most important are the functions that allow the ship to fire and move.

```
// Get input and clamp the players movement within a set boundary
float xPos = transform.position.x + (Input.GetAxis("Horizontal") * playerSpeed);
float yPos = transform.position.y + (Input.GetAxis("Vertical") * playerSpeed);
playerPos = new Vector2(Mathf.Clamp(xPos, xmin, xmax), Mathf.Clamp(yPos, ymin, ymax));

// Move player
transform.position = playerPos;
```

Laser Defender Player Movement Code

```
if (Input.GetKey(KeyCode.Space) && Time.time > nextFire)
{
    // Set the delay to prevent player from firing again until projectile delay is over
    nextFire = Time.time + projectileDelay;

    // Create the projectile and give it a velocity
    projectilePosition = new Vector3(playerPos.x, playerPos.y);
    GameObject projectileInstance = Instantiate(projectile, this.playerPos, Quaternion.identity) as GameObject;
    projectileInstance.GetComponent<Rigidbody2D>().velocity = new Vector3(0, projectileSpeed);
    AudioSource.PlayClipAtPoint(fireSound, transform.position);
}
```

Laser Defender Player Fire Code

The player movement code is within a function that is repeatedly called as long as the game is running. It registers input on the left and right arrow keys as a negative or positive value in Input.GetAxis(“Horizontal”) and the same for up and down in Input.GetAxis(“Vertical”). The

function then multiplies these values by the player's speed, a static value that determines how fast the player moves. This multiplied value is then clamped to make sure the player does not leave the screen and is then made the player's position.

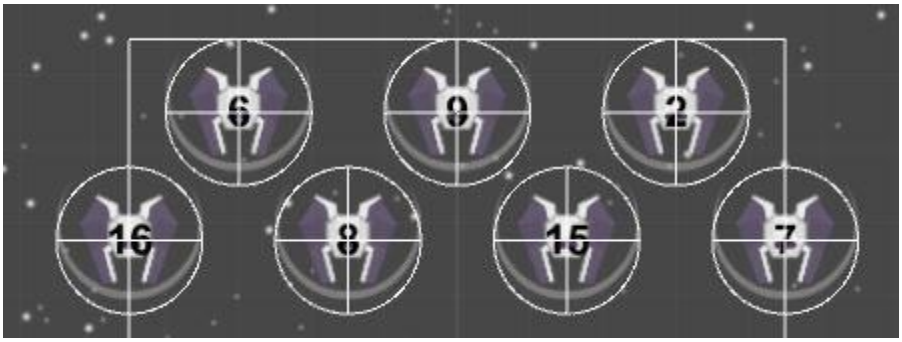
The player fire code works by listening for the space bar to be pressed and checking to make sure that the ship has not fired within the next Fire period; this allows a forced delay between shots so the player cannot fire continuously. Once these conditions are met, the delay is reset, and the projectile is spawned and given a forward velocity. A sound is also played as the projectile is fired.

C. Enemy Spawning script

```
void SpawnEnemies()
{
    //spawn all the enemies
    foreach (Transform child in transform)
    {
        GameObject enemy = Instantiate(enemyPrefab, child.transform.position, Quaternion.identity) as GameObject;
        enemy.transform.parent = child;
    }

    // Set the number of each enemey
    SetNumbersToEachEnemy();

    // Set an initial Laser Power for the wave
    GameManager.instance.CalculateLaserPower();

}
```

Laser Defender Enemy Spawn Code

The spawning script will take the formation object and populate it with enemies at positions set within the object. It then runs the set number to each enemy function that goes through each enemy, in each position, and populates it with a number.



Laser Defender Enemy Spawn Code

As can be seen in the image above, the box roughly represents the formation's position while each circle represents the position a ship will be stored. The code that places the numbers on the various ships is also shown below.

```
// Sets a random number to each enemy between a specified minimum and a specified maximum
void SetNumbersToEachEnemy()
{
    // go into each position in the formation
    foreach (Transform child in transform)
    {
        // go into each enemy in a position
        foreach (Transform grandChild in child)
        {
            //go into the text mesh object attached to the enemy
            foreach (Transform greatGrandChild in grandChild)
            {
                // Check to make sure enemy does not already have a number
                if (greatGrandChild.GetComponent<TextMesh>().text.Length == 0)
                {
                    // Add number to enemy
                    int randomValue = Random.Range(minShipValue, maxShipValue);
                    greatGrandChild.GetComponent<TextMesh>().text = randomValue.ToString();
                    while (!EnemyHash.Add(randomValue))
                    {
                        randomValue = Random.Range(minShipValue, maxShipValue);
                        greatGrandChild.GetComponent<TextMesh>().text = randomValue.ToString();
                    }

                }
            }
        }
    }
}
```

Laser Defender Code that gives every enemy ship a number

This code works by drilling into the enemy formation and then drilling into each position within that formation and populating the text object attached to that position with a number.

Laser Defender 2nd Version Final Prototype

The final prototype version improved on every single screen from the first prototype and included the options screen.

1. Main Menu Screen

Laser Defender V2 Main Menu

The two main changes in this final prototype compared to the original main menu screen are the addition of the star field background to make it more visually appealing. The second change was the addition of the options button to allow the player to access the options menu.

2. Options Menu

Laser Defender V2 Options Menu

The above image shows the options menu; it allows you to select one of four different equation types. These four equation types are addition, subtraction, division, and multiplication. It also allows the player to select the minimum possible and maximum possible operand values that will be generated for the selected equation type. Hitting back brings you back to the main menu.

3. Play Screen

Laser Defender V2 Play Screen

The main change in the play screen in the final prototype is the addition of different enemy sprites so that every spawn will have a different looking kind of enemy. In addition to the new enemy types, enemies can now drop two different types of power-ups. The first power-up is a shield that will heal the player 100 points of health. Custom fire, explosion, and power-up sound effects were also added. The second power-up is a star that will give the answer to the current equation. Otherwise, the screen works exactly the same as in the previous version.

4. Game Over Screen

Laser Defender V2 Game Over Screen

The final prototype version of the game over screen is identical to the previous version except for the star field background to make it feel a little less dull.

5. Interesting bits of code

A. Complete Redesign, Overhaul, and Expansion of the Equation Generation script

This final equation generation script uses four different functions to generate random equations for addition, subtraction, multiplication, and division problems. Below we shall look at each of the four functions and then discuss their function in detail.

```
public void GenerateListOfEnemyAnswersAndEquationsAddition(int formationSize)
{
    // Variable Definitions
    int randomValue1 = 0;
    int randomValue2 = 0;
    int answer = 0;
    string equation;
    bool duplicateFound = false;
    EnemyList = new List<float>();
    EquationsList = new List<string>();

    // Equation Generation Loop
    for (int i = 0; i < formationSize; i++)
    {
        randomValue1 = Random.Range(MinOperandValue, MaxOperandValue);
        randomValue2 = Random.Range(MinOperandValue, MaxOperandValue);
        answer = randomValue1 + randomValue2;
        equation = randomValue1 + " + " + randomValue2 + " = ?";
        if (!((MaxOperandValue - MinOperandValue * 2) <= formationSize))
        {
            // Check if number is already in list
            for (int j = 0; j < EnemyList.Count; j++)
            {
                if (EnemyList.Contains(answer))
                    duplicateFound = true;
            }
        }

        // Check duplicate flag if true generate a new number else add number to list
        if (duplicateFound)
        {
            i--;
            duplicateFound = false;
        }
        else
        {
            EnemyList.Add(answer);
            EquationsList.Add(equation);
        }

    }

}
```

Laser Defender V2 Addition Function

```
public void GenerateListOfEnemyAnswersAndEquationsSubtraction(int formationSize)
{
    // Variable Definitions
    int randomValue1 = 0;
    int randomValue2 = 0;
    int answer = 0;
    string equation;
    bool duplicateFound = false;
    EnemyList = new List<float>();
    EquationsList = new List<string>();

    // Equation Generation Loop
    for (int i = 0; i < formationSize; i++)
    {
        randomValue1 = Random.Range(MinOperandValue, MaxOperandValue + 1);
        randomValue2 = Random.Range(MinOperandValue, randomValue1 + 1);
        answer = randomValue1 - randomValue2;
        equation = randomValue1 + " - " + randomValue2 + " = ?";
        if (!(MaxOperandValue - MinOperandValue <= formationSize))
        {
            // Check if number is already in list
            for (int j = 0; j < EnemyList.Count; j++)
            {
                if (EnemyList.Contains(answer))
                    duplicateFound = true;
            }
        }

        // Check duplicate flag if true generate a new number else add number to list
        if (duplicateFound)
        {
            i--;
            duplicateFound = false;
        }
        else
        {
            EnemyList.Add(answer);
            EquationsList.Add(equation);
        }

    }

}
```

Laser Defender V2 Subtraction Function

```
public void GenerateListOfEnemyAnswersAndEquationsMultiplication(int formationSize)
{
    // Variable Definitions
    int randomValue1 = 0;
    int randomValue2 = 0;
    int answer = 0;
    string equation;
    bool duplicateFound = false;
    EnemyList = new List<float>();
    EquationsList = new List<string>();

    // Equation Generation Loop
    for (int i = 0; i < formationSize; i++)
    {
        randomValue1 = Random.Range(MinOperandValue, MaxOperandValue);
        randomValue2 = Random.Range(MinOperandValue, MaxOperandValue);
        answer = randomValue1 * randomValue2;
        equation = randomValue1  + " * " + randomValue2 + " = ?";

        if (!(((MaxOperandValue - MinOperandValue) * (MaxOperandValue - MinOperandValue)) <= formationSize))
        {
            // Check if number is already in list
            for (int j = 0; j < EnemyList.Count; j++)
            {
                if (EnemyList.Contains(answer))
                    duplicateFound = true;
            }
        }

        // Check duplicate flag if true generate a new number else add number to list
        if (duplicateFound)
        {
            i--;
            duplicateFound = false;
        }
        else
        {
            EnemyList.Add(answer);
            EquationsList.Add(equation);
        }

    }

}
```

Laser Defender V2 Multiplication Function

```
public void GenerateListOfEnemyAnswersAndEquationsDivision(int formationSize)
{
    // Variable Definitions
    int randomValue1 = 0;
    int randomValue2 = 0;
    int answer = 0;
    string equation;
    bool duplicateFound = false;
    EnemyList = new List<float>();
    EquationsList = new List<string>();

    // Equation Generation Loop
    for (int i = 0; i < formationSize; i++)
    {
        randomValue1 = Random.Range(MinOperandValue, MaxOperandValue);
        if(randomValue1 == 0)
        { randomValue1 = 1; }

        randomValue2 = Random.Range(MinOperandValue, MaxOperandValue);
        if (randomValue2 == 0)
        { randomValue2 = 1; }

        answer = randomValue2;
        equation = randomValue1 * randomValue2 + " / " + randomValue1 + " = ?";

        if (!(MaxOperandValue - MinOperandValue <= formationSize))
        {
            // Check if number is already in list
            for (int j = 0; j < EnemyList.Count; j++)
            {
                if (EnemyList.Contains(answer))
                    duplicateFound = true;
            }
        }

        // Check duplicate flag if true generate a new number else add number to list
        if (duplicateFound)
        {
            i--;
            duplicateFound = false;
        }
        else
        {
            EnemyList.Add(answer);
            EquationsList.Add(equation);
        }

    }
```

Laser Defender V2 Division Function

All of the equation generation functions follow a simple pattern, they take in an integer, that integer is the number of ships in the formation that is requesting the equations. They then generate two random values that are transformed into an equation. Each function also checks if the answer is already on the list to ensure that each ship has a unique number. Each function additionally checks that the formation is large enough to support a unique number for each ship, if not this requirement is dropped. Once the equation is generated and determined not to be a duplicate, the equation and answer are placed on the list to be referenced by other functions. The addition, subtraction, and multiplication functions all follow an identical pattern. They all generate two operands, perform the requested mathematical operation on these two operands storing the result as an answer, then construct a string using the two values to create the equation. The division function works a bit differently first it has a check when generating random numbers to turn any zero results into a one to avoid a division by zero. Second when it generates the two values, the dividend is the multiplication of the first and second values, and the divisor is the first value. The answer is then simply the second value, thus generating a division equation that does not require any actual division operations.

Conclusion

From beginning to end this project was a passion project to create a functional prototype for a game I hope to expand upon in the future. While in its current form it is extremely basic, this project has potential, and the core mechanic of a math based shooter appears to be viable. The core mechanics for an expanded game are all in place, but to make this a complete game and not simply a gameplay prototype would require a lot of work and additions in animation, art, more levels, custom sound, and much more. Not to mention from a coding standpoint it is likely that a nearly

complete restructuring and redesign of the source code would be necessary. Mainly because the way in which the prototype was designed is not very conducive to scalability. The prototype shows the idea has merit and could work if it was a larger scale project. I plan to continue to refine and work on the idea and create a more fully featured alpha over the next year to explore if this gameplay idea can scale to fill an entire game. Regardless if that is successful or not working on this project has been an eye-opening experience when it comes to the difficulties and hurdles of game development. Even on this relatively small project with the advantage of having a professional game engine, there were quite a few snags and challenges that caught me by surprise. In the end, I was able to persevere and finish this prototype and will take the lessons I learned onto all my future projects, game-related or otherwise.