Journal of Cybersecurity Education, Research and Practice

Volume 2020 | Number 1 Article 2

June 2020

MalAware Defensive: A Game to Train Users to Combat Malware

Tyler Moon
University of North Georgia, tcmoon3528@ung.edu

Tamirat Abegaz
University of North Georgia, tamirat.abegaz@ung.edu

Bryson Payne
University of North Georgia, bryson.payne@ung.edu

Abi Salimi
University of North Georgia, abi.salimi@ung.edu

MalAware Defensive: A Game to Train Users to Combat Malware

Abstract

Several research findings indicate that basic cyber hygiene can potentially deter the majority of cyber threats. One of the ways cybersecurity professionals can prepare users to ensure proper hygiene is to help them develop their ability to spot the difference between normal and abnormal behavior in a computer system. Malware disrupts the normal behavior of a computer system. The lack of appropriate user training has been one of the main reasons behind the exposure of computer systems to threats, from social engineering to viruses, trojans, and ransomware. Basic knowledge about common behavioral characteristics of malware could help users identify potentially abnormal behavior in the systems they use on a daily basis.

Games with a purpose beyond entertainment are becoming an integral part of educational training. This is even more relevant to the field of cybersecurity, where there are many threat agents targeting individuals and organizations. The purpose of this paper is to describe a game, MalAware Defensive, developed to increase users' awareness of common malware behaviors and their impact on a system, as well as to explain ways to combat various major types of malware. The game's design is based on research showing that content disseminated in an interactive game provides a lasting impact on the retention of concepts. The game provides relevant knowledge about various types of malware, the behavior and impact of malware on a computer system, and several ways to avoid infection and compromise. The game, through its interactive gameplay environment, with rewards for answering questions correctly, could potentially help players improve their understanding of malware, how to detect its presence, and how to defend against it.

Keywords

cybersecurity, cyber hygiene, security training, malware awareness

INTRODUCTION

Malware has been an ever-present and growing issue and a pervasive threat to both individual users and businesses. Basic knowledge about malware could help the average user identify potential abnormal behavior in a system caused by malware. Advances in game engines have facilitated the creation of cost-effective systems for game-based interactive training in various domains.

Educational games provide a player with not only an entertaining experience but also an interactive platform for learning educational content. As technology has become ubiquitous in our lives, so have malicious attacks on technology-based systems. Malware is intentionally malicious software which can harm a computer system in various ways. In many cases, malware finds its way into a computer system by an inadvertent action of a user. Training users on how to recognize and appropriately react to malware can decrease possible damage to computer systems.

The purpose of this paper is to describe a game, MalAware Defensive, designed to increase awareness of malware by understanding its behavior on a system, as well as to explain ways to combat major types of malware. It is based on the emphasis that teaching materials disseminated in an interactive game have been shown to provide a lasting impact on the retention of concepts. The game is designed for ages 12 and up, specifically for individuals with limited prior security training, and provides informative knowledge about various categories of malware, the impact and behavior of malware on a system, and best practices for avoiding malware. The game, through its interactive gameplay environment and rewards for answering questions correctly, could potentially help players to understand the behavior of malware. We have designed and developed MalAware Defensive as a tower-defense gaming system, to teach users about different types of malware and how to avoid them.

RELATED WORK

Games with a purpose beyond entertainment are becoming an integral part of educating and training users and employees. This is even more relevant to the field of cybersecurity, where there are many common threats and users may not be aware of their existence. As Guimaraes and colleagues described in their work, educational software for security is important since traditional training has come up short as the main source of learning threat categories and defense mechanisms (Guimaraes et al., 2012). A so-called serious game allows a user to learn security concepts in a hands-on and entertaining learning environment.

There are several serious games within the security education genre. Anti-Phishing Phil (Guimaraes et al., 2012) teaches a player to identify URLs that are

phishing attempts. As the player clicks on the safe and fake website URLs, the game explains why a URL was safe or dangerous to teach the key concepts of avoiding phishing attempts. Auction Hero (Chiasson et al., 2011) takes it a step further by presenting a series of missions that teach a player different computer security concepts such as weak passwords, malware, and phishing. Role-playing scenarios within the game make security skills and knowledge integral to beating each level.

The design of such educational security games follows a more rigorous process than the ones for the games created for pure entertainment. Nagarajan et al. (2012) explain that serious games must not only use a game dynamic, which is entertaining, but it must also reward the player for making correct choices. This will encourage a player to continue playing and learning. Since there are many forms of games (e.g., survival games, strategy games, etc.), educational games must follow a developmental process different from that of traditional video games. Larsen and Majgaard (2016) describe the 'expanded game design space' process that their students use to create educational computer games. The process starts with the basic steps of choosing the game's purpose and mechanics and then moves on to game balancing techniques, reward structures, game feel, and feedback mechanisms to aid a player's learning experience. This process serves as the basis of this project's design approach.

Use of serious games in security training is relatively new. There are not many studies on this matter, however, Hendrix et al. (Hendrix et al., 2016) have studied 28 papers on security training games. They found that only two games had positive effects on students' retention of security concepts. This indicates both the immaturity of this field of study and difficulty in evaluating the effectiveness of the serious game platforms compared with traditional learning approaches. The aim of this paper and the developed game is to aid in learning malware concepts and to add another serious game to the security training subgenre.

DEFINITION OF MALWARE

Malware (short for **mal**icious soft**ware**) refers to any software that causes intentional harm. Its purpose could be for financial gain, to harm the hard drive, to steal users' confidential information, to turn a machine into a bot or to make a computer unavailable to the user (Singh, J. 2018). Below are the most common malware types which are explored in the MalAware Defensive game.

Virus

A virus, probably the most common type of malware, depends on an existing, non-dangerous program. It binds itself to an executable. When the executable is run, the virus spreads. For instance, after downloading an infected file from the internet, the

file must be opened to allow the attached virus to cause harm.

Worm

Worms vs. viruses are stand-alone programs. They do not need to attach to anything else. They are also capable of self-replication. That is, they can spread without user interaction. Worms can slow a computer down immensely in a short period of time, because of their quick spread.

Trojan Horse

A Trojan horse is any malware that disguises itself in a legitimate and non-threatening file or program. It is even able to bind to non-executables such as images and audio files.

Spyware

Spyware is a malware that moves stealthily into a computer system and steals personal information. It can track user activities on the internet and collect personal information such as usernames, passwords, and banking information.

DESIGN

Concepts and Terminologies

Standard **tower defense games** are strategy games, where various types of towers (the defense weapons) are placed into the environment to defend against a wave of attackers attempting to make it to the end of a course. There are typically various types of towers, each with their own set of powers and abilities. There are also different types of enemies, usually with different amounts of health and strengths/weaknesses against certain turrets. A player starts with a set quantity of lives and money and usually gains money from completing waves successfully and/or by eliminating enemies. The player must manage their money and place their towers strategically because certain enemies are sometimes immune to particular towers or attacks. For example, in the famous tower defense game, Bloons TD, camouflage balloons are immune to damage unless a tower can detect them. Because of these game mechanics, we developed a tower defense game to create an entertaining serious game environment to teach cybersecurity concepts.

Serious games, compared to standard tower defense games, require additional game creation concepts to make them educational. A standard tower defense game teaches strategy at a basic level. The gameplay does not usually intertwine with everyday concepts. Most of these games are fantasy or military-themed. Serious games share not only the standard theme of being entertaining, but also serving as

a strong reward structure to motivate the players to continue playing and learning. This project used the six basic elements of serious game design: game objects, game-balancing, reward structure, game-feel, communication flow, and feedback.

Architectural Design

MalAware Defensive is a tower defense game at its core, but with the aspects of a serious game. In its design, educational contents are interconnected with the gameplay, making them reliant on one another. The game is created in Unity 3D, a cross-platform game engine that is free for developers. In Unity 3D, scenes, which act as the "stages" for the game, can be created. MalAware Defensive consists of three scenes: the *main menu scene*, which allows the player to either play or close the game, the *tip scene* that consists of the panels telling the player about the enemies and turrets, and the *main level scene*, where the tower defense game is played. Figure 1 shows the scenes, the scene interactions, and each scene's main game components.

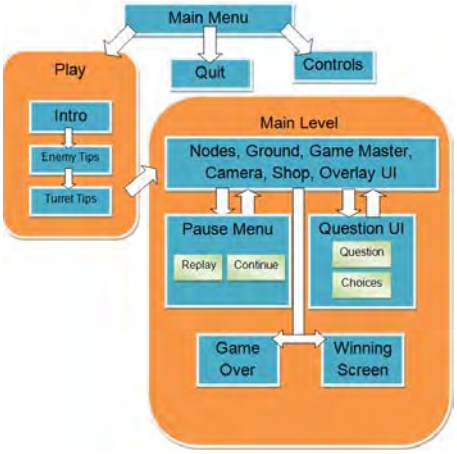

Figure 1. Depiction of the architecture within Unity

The communication flow between the various game components must be streamlined. An important aspect of the architectural design of a video game is that a change to code must be easily propagated across all the respective moving pieces. For example, enemies in a tower defense game have their own models and states, created from the same Enemy script. As with the turrets, each turret is vastly different from the next. Each turret uses the same Turret script to assign its states and a blueprint script that allows easy modification of its prices/sell values in the shop. Adding or changing objects becomes a trivial task once the basic architecture of each element is established.

The hub of communication flow in MalAware Defensive is a game object called Game Master. As shown in Figure 2, this is an empty object that holds every main

game script and allows them to be modified in the Unity inspector. That is, player states can be updated, the amount and types of enemies can be adjusted for the wave spawner, the number/content of quiz questions can be updated, and so on.

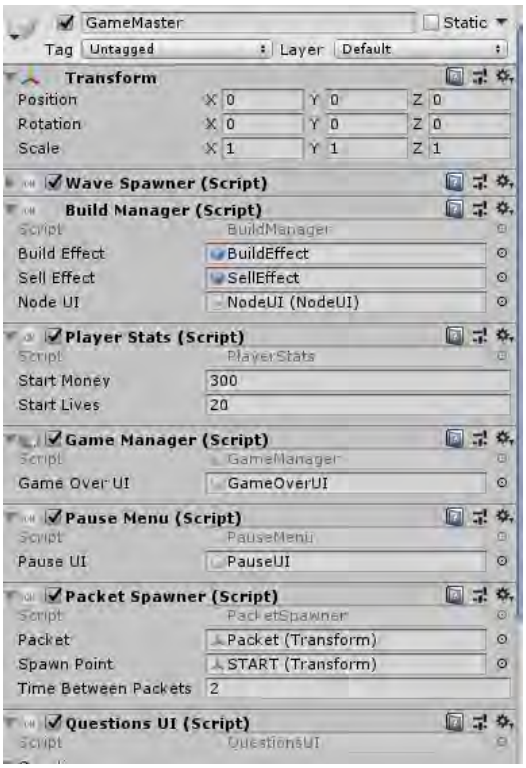

Figure 2. The Game Master within the inspector

User Interface Design

The UI components of MalAware Defensive allow a player to interact with the environment and play the game. The buttons in all the menus are either animated or change color when they are hovered over.

The game opens with a menu allowing a player to start a game (Figure 3).

Figure 3. The Main Menu

An introduction explains the premise of the game and provides tips for each type of malware and turret. Within the main level, Figure 4, the player is presented with a gridded layout of nodes. A green block represents the enemy's start position, and a red block represents the enemy's end position. Across the bottom of the screen, there are the icons of the turrets that the player can select, and the amount of money the player has. Hovering over a turret reveals information about the tower, and once pressed, the player can click any of the squares on the map to place it. If the square appears grey, the player can place the turret. However, if the square is red, the player has run out of funds. Clicking on a placed turret also reveals upgrade and sell options.

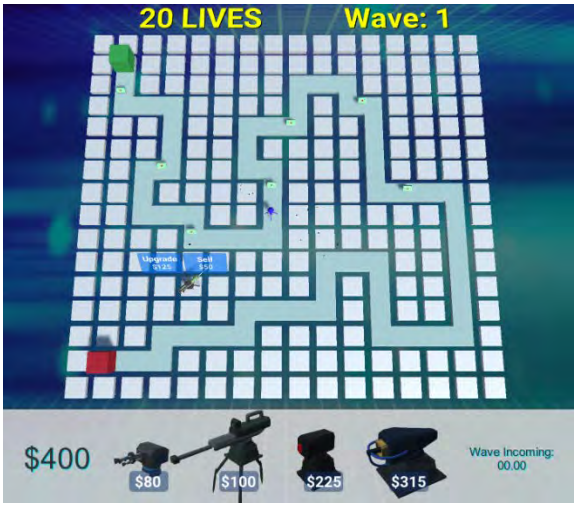

Figure 4. The player's view of the Main Level

The stage represents a connection, where the starting point is the open Internet and the endpoint is a host machine. The goal is to stop the malware from making it

to the host. Green “packets”, which are constantly running through the course represent the legitimate files and traffic flowing through the computer.

Answering malware-related cybersecurity questions is the main aspect of education in this game. The game pops up quiz questions between rounds (Figure 5). When the player answers a question correctly, he or she will receive a reward such as more money to spend on turrets. If a question is answered incorrectly, a negative effect, such as losing money or the enemies gaining a buff, will take place. For the players with no background knowledge, the game provides tips and information on the enemies and turrets before the game begins. The first several rounds of a game will introduce the different enemies and respective strengths and weaknesses and how to stop them.

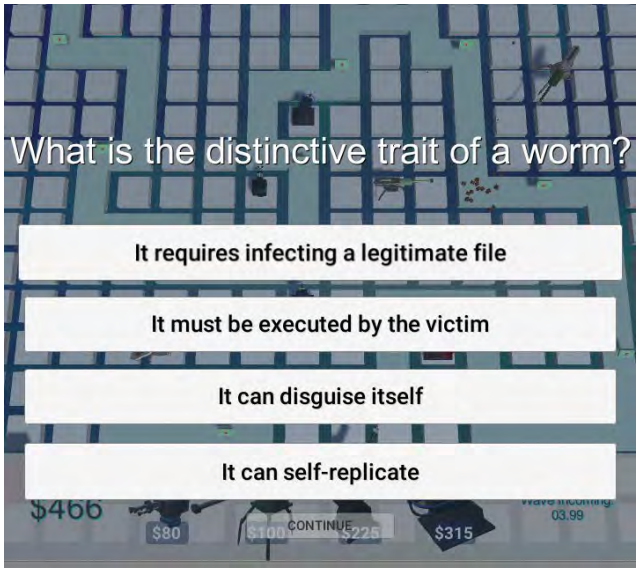

Figure 5. The Quiz UI

Enemy Design

Each enemy represents a different type of malware. There are viruses, worms, Trojan horses, and spyware to be eliminated. There are also other types of malware attacks such as ransomware that are explored throughout the game.

To make it fun and easy to remember, the malware enemies are modeled after their respective correlating objects, which are created by using Blender 3D software -- Viruses are modeled after bacteriophage, worms are modeled after a worm, Trojan horses are represented by a wooden horse (Figure 6), and spyware is represented by a masked figure.

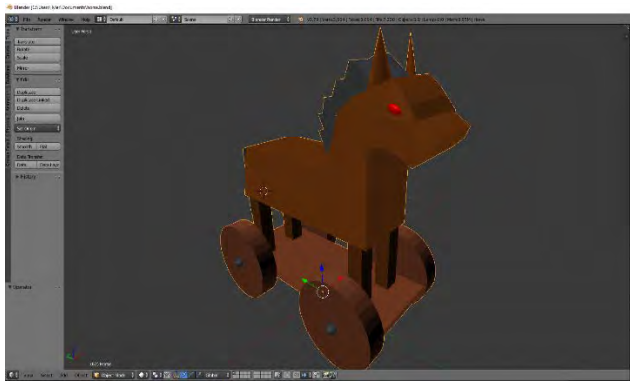

Figure 6. The Trojan Horse model in Blender

Design of enemies led to the game-balancing step in development. Each malware enemy had to have varying speeds, health pools, and possibly abilities to match their characteristics. This would directly affect the value that each of the turrets could bring to the player, as certain turrets would be better against certain enemies—viruses are common, so they were given moderate health and speed. Worms are slower, tougher to destroy and can self-replicate. Trojan horses are concealed, hard to destroy and slow. Spyware is concealed, has low health, and is quick. Each enemy brings a unique challenge for the player to overcome.

Turret Design

Anti-malware software is a tool for combating malware. Turrets act as the arsenal of weapons used by a player acting as anti-malware software. Each turret has a purpose such as being able to cause reliable damage, high damage and slowing effect, or it can detect Trojan horses and spyware. The turrets are designed in Blender, imported into Unity, and linked to the turret- and bullet scripts.

The game offers the following turrets: **Destroyer**, which causes moderate damage and has a moderate fire rate. It causes the most reliable form of damage. In the context of anti-malware software, Destroyer seeks and destroys the more easily found types of malware. **Shredder**, which causes high damage and has a low fire rate. It is the only turret capable of damaging multiple targets at once using its area of effect (AOE) mechanic. Shredder represents the anti-malware's ability to shred multiple files quickly and completely from a computer's hard drive. **Quarantine** turret is the only turret that shoots a constant beam of energy at an enemy. It slows "debuffs" the enemy preventing it from moving at its full speed. The beam deals low damage but gives more time to other turrets to destroy the target. Quarantine represents the ability of anti-malware software to quarantine a threatening file or application so it cannot cause more damage. The last turret is **Scanner**, which does low damage and has a high fire rate. Scanner, the only non-upgraded turret, can

detect concealed malware (e.g., Trojan horses and the spyware). Scanner represents the Anti-malware's ability to live scan for threats on a network and try to eliminate them in real-time. It, essentially, "pings" the threat with the low damage while cutting down their health over time.

IMPLEMENTATION

The MalAware software consists of 25 scripts written in C#. The following sections will cover the most important aspects of key classes.

Enemies

Two scripts give the malware enemy-models life. The **Enemy script** contains the variables for the health, movement speed, value and concealment status, and the methods for taking damage and dying. When an enemy dies, it is destroyed from the scene and the player gains money (Figure 7).

```
public void takeDamage(float amount)
{
   health -= amount;

   if(health <= 0)
   {
      Die();
   }
}
```

```
void Die()
{
   GameObject effect = Instantiate(deathEffect, transform.position, Quaternion.identity);
   Destroy(effect, 5f);
   Destroy(gameObject);
   PlayerStats.money += worth;
}
```

Figure 7. takeDamage() and Die() methods

The second script is **Enemy Movement**, which controls how an enemy moves through the map. A separate waypoint script tracks the invisible checkpoints at each turn on the map. Once spawned, the enemy uses the array of waypoints that are set at each turn on the track. While the enemy is moving, the movement script's Update method is constantly checking to see where the enemy is on the map. The quaternion and vector data types allow the enemy to turn to a new direction at each waypoint and constantly move through the level (Figure 8):

```
Vector3 direction = target.position - transform.position;

Quaternion rotation = Quaternion.LookRotation(direction, Vector3.up);
transform.rotation = rotation;

transform.Translate(direction.normalized * enemy.speed * Time.deltaTime, Space.World);

if (Vector3.Distance(transform.position, target.position) <= 0.4f)
   getNextWaypoint();

enemy.speed = enemy.startSpeed;
```

Figure 8. Enemy Movement

Turrets

The turrets are controlled by the main Turret script, and directly interact with the Bullet, Node, Turret Blueprint, Shop, Build Manager, and the Enemy scripts. The turret variables include the range, type of bullet, if it uses a laser, and if it can shoot concealed malware. There are methods for updating the target and shooting the enemy. The most important method locks on the new target. By using quaternions, the top part of the turret can be rotated to face the enemy in real-time (Figure 9).

```
  void lockOnTarget()
  {
     Vector3 dir = target.position - transform.position;
     Quaternion lookRotation = Quaternion.LookRotation(dir);
     Vector3 rotation = Quaternion.Lerp(partToRotate.rotation, lookRotation, Time.deltaTime *
turnSpeed).eulerAngles;
     partToRotate.rotation = Quaternion.Euler(0f, rotation.y, 0f);
  }
```

Figure 9. lockOnTarge Method

The Bullet script controls what happens when the target is hit and whether an explosion would take place to cause area damage (the Shredder's ability). The Node, Turret Blueprint, Shop, and Build Manager scripts control whether a turret has been placed within a certain node of the map. If it has not, the turret is bought and placed. If it has, then clicking the turret brings up the upgrade- and sell menu within the Node UI script.

Spawning Waves

The Wave Spawner script holds enumerators and an internal class of waves (Figure 10).

```
public enum SpawnState { SPAWNING, WAITING, COUNTING }

[System.Serializable]
public class Wave
{
   public string name;
   public Transform[] enemy;
   public int count;
   public float rate;
}

public Wave[] waves;
```

Figure 10. The Wave Spawner Script

The 'System.Serializable' tag allows all variables within the Wave class to be changed within the Unity inspector. This allows the Game Master to set the number and types of enemies, which will spawn, and add as many customized rounds as deem fit.

The IEnumerator class is used to spawn enemies within a wave at a given rate. This allows enemies to spawn, spaced out from each other. It must be called within the StartCoroutine method by typing

StartCoroutine(SpawnWave(waves[nextWave]));

and using the methods shown in (Figure 11):

```
IEnumerator SpawnWave(Wave _wave)
{
   Debug.Log("Spawning Wave: " + _wave.name);
   state = SpawnState.SPAWNING;

   for (int i = 0; i < _wave.count; i ++)
   {
      int randomEnemy = Random.Range(0, _wave.enemy.Length);
      SpawnEnemy(_wave.enemy[randomEnemy]);
      yield return new WaitForSeconds(1f / _wave.rate);
   }

   state = SpawnState.WAITING;

   yield break;
}

void SpawnEnemy(Transform _enemy)
{
   Debug.Log("Spawning Enemy: " + _enemy.name);
   Instantiate(_enemy, spawnPoint.position, spawnPoint.rotation);
}
```

Figure 11. SpawnWave Method

As it is shown above, the enemy spawned in a round is determined by a random number within the bounds of the enemy array for that specific wave.

Quiz Questions

A malware quiz occurs every other round and is controlled by the QuestionsUI script. In addition to the button methods, there is a serializable Question class, which allows total customization. That is, questions can be added or updated in Game Master. The game starts with a private list of unanswered questions. The GetRandomQuestion method is called when a round divisible by 2 is reached. This method removes a randomly selected question from the list (Figure 12).

```
void GetRandomQuestion()
{
   int randomIndex = Random.Range(0, unansweredQuestions.Count);
   currentQuestion = unansweredQuestions[randomIndex];

   unansweredQuestions.RemoveAt(randomIndex);

   questionText.text = currentQuestion.question;

   if(currentQuestion.trueFalse == true)
   {
      b1.interactable = false;
      b4.interactable = false;
   }

   choice1.text = currentQuestion.choice1;
   choice2.text = currentQuestion.choice2;
   choice3.text = currentQuestion.choice3;
   choice4.text = currentQuestion.choice4;

   funFact.text = currentQuestion.funfact;
}
```

Figure 12. GetRandomQuestion Method

After the player clicks on an answer, the OnClick event code of the button updates the screen. The player will be given an indication of whether they answered the question correctly. Money is either added or subtracted from their funds, and a helpful tip about the question is displayed at the top of the screen (Figure 13).

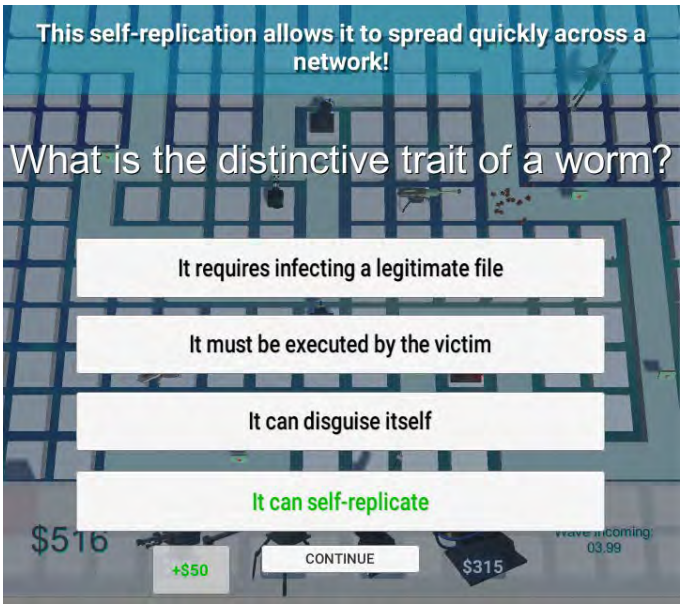

Figure 13. Once answered, the UI gives feedback to the player

Game States

The game consists of a main menu, tip menu, pause menu, question menu, game over menu, and a winning menu. For these game states to function correctly, each UI script must refer to the UI GameObject in the Unity inspector to allow it to be updated through the code.

To implement the pause menu, the Update method checks if the 'P' or 'ESC' key is pressed. If so, the Update method calls the toggle method to pause the game. This method sets the UI to the opposite state of it is currently in. The same toggle method is used by the question UI script, since the game must be frozen while the player answers a question (Figure 14).

```
public void Toggle()
{
   questionUI.SetActive(!questionUI.activeSelf);

   if (questionUI.activeSelf)
   {
      Time.timeScale = 0f;
   }
   else
   {
      Time.timeScale = 1f;
   }
}
```

Figure 14. The Toggle Method

Answering a quiz question state is activated when the end of an even wave is reached. This is done in the wave spawning script, which determines if the current completed wave is divisible by 2. If so, the Player's static Boolean *answeringQuestion* is set to true, and the UI is toggled on. After the question is answered, the *continue* button can be pressed to toggle the UI back and set the game to the *play* state.

As for winning or losing the game, a Game Manager script constantly checks for certain events. To lose the game, the player's health must reach zero or the "e" key must be pressed. To win the game, the player must complete all rounds (Figure 15 and Figure 16).

Figure 15. The Game Over screen displays the waves survived and allows the player to try again or go to the menu.

Figure 16. The game is won upon completing 50 waves

To make a menu button functional, its corresponding method must be linked to it through the Unity inspector. This is done by adding an OnClick event to the button and choosing the corresponding method. Unity uses a SceneManager object to change game states. The methods of the menu and replay buttons change the scenes. To start a game over, the scene manager must reload the current scene with *SceneManager.GetActiveScene().name*. To change to the main menu, the name of the scene is passed to the method, instead.

RESULTS & CONCLUSIONS

The MalAware Defensive design and implementation have led to the creation of a unique tower defense game that has aspects of a serious game. The designed UI makes the game easy to understand for first-time players. The cybersecurity information about malware is informative but not overbearing. MalAware Defensive provides the player with an awareness of the malware threats. Displaying these concepts in an entertaining way and allowing a player to build an association with in-game enemy models provide an interactive learning environment that traditional methods cannot provide.

While the effectiveness of this game in awareness of malware concepts is not fully investigated, test cases with the players who had no background knowledge of malware threats showed a promise on the retention of information after the game was over.

By following the principles described in the game design literature, namely core mechanics, game-feel, flow, reward structure, and game objects, we developed a game which is both entertaining and educational. Game balancing is still an ongoing effort due to the nature of the game. The implementation through object-oriented programming in C# allowed for a highly structured and easy-to-understand coding process. Paired with the Unity interface and Blender modeling software, the game creation became an enjoyable first-time experience as an undergraduate senior research project.

As a part of our future work, we would like to incorporate more Malware including social engineering-related training and practices and also to determine level of knowledge retention among a larger number of users.

REFERENCES

Chiasson, S., Modi, M., Biddle, R. (2011). Auction Hero: The Design of a Game to Learn and Teach about Computer Security. In C. Ho & M. Lin (Eds.), *Proceedings of E-Learn 2011--World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education* (pp. 2201-2206). Honolulu, Hawaii, USA: Association for the Advancement of Computing in Education (AACE). https://www.learntechlib.org/primary/p/39053/.

Guimaraes, M., Said, H., Austin, R. (January 2012). Experience with Videogames for Security. *The Journal of Computing Sciences in Colleges* 27, 3, 95–104. DOI:http://dx.doi.org/10.4324/9781315622743-4

Hendrix, M., Al-Sherbaz, A., Bloom, V. (2016). Game Based Cybersecurity Training: are Serious Games suitable for cybersecurity training? *International Journal of Serious Games* 3, 1 (2016), 1–10. DOI:http://dx.doi.org/10.17083/ijsg.v3i1.107

Larsen, L. J., Majgaard, G. (2016). Expanding the Game Design Space – Teaching Computer Game Design in Higher Education. *Designs for Learning*, *8*(1), 13–22. DOI:http://dx.doi.org/10.16993/dfl.68

Nagarajan, A., Allbeck, J.M., Sood, A., Janssen, T.L. (2012). Exploring game design for cybersecurity training. *2012 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)* (2012). DOI:http://dx.doi.org/10.1109/cyber.2012.6392562

Singh, J. (2018). Challenges of Malware Analysis: Obfuscation Techniques. *International Journal of Information Security Science* 7, 3, 100–110. http://search.ebscohost.com.libproxy.ung.edu/login.aspx?direct=true&db=a9h&AN=133550405&site=eds-live&scope=site