No Drone Zone: Tower Defense Game

June 09, 2017

Perseus Team:
Steven Ha
Thomas Mathew
Gabriel Orellana

I. Introduction:

For its magnum opus, Team Perseus set out to make a Tower Defense game reflective of the times. Privacy has become a large concern. Not only do people have to worry about the government reading all their emails and tracking our fitbits, they also have to worry about their nosey neighbors buzzing over our houses with easy to purchase drones. To help alleviate some of the stress which is so prevalent, Team Perseus made it their mission to allow people to fight back, at least symbolically. No Drone Zone is a game which pits the player against the evil forces of privacy invasion in a fun, but also challenging, arena. In order to bring this game to fruition, Team Perseus struggled through an initially steep learning curve, spending countless hours learning the art of Unity and programming in C#. The Team tapped into their artistic sides, producing state of the art sprites, hand crafted in MS Paint. To further challenge themselves, Team Perseus added mySQL functionality, allowing the user to store their precious game data so that they could return later and resume taking out their frustrations.

The final result of all this hard work is a unique Tower Defense game where the player battles drones of all shapes and sizes, utilizing a variety of towers to fight the seemingly endless onslaught. If the player is cunning and strong, they can find satisfaction in successfully defending their parks, neighborhoods, and ultimately, the world from the threat of constant surveillance!

II. Game Description:

'No Drone Zone' is a 2D tower defense game that was designed to be played on the internet via the Google Chrome web browser. The idea behind the game is basically someone trying to get some peace and quiet in their home, but constantly being disturbed by drones. To combat the drones, they put up towers that attempt to shoot the drones down before they can make it to their targets. The user begins with a set amount of money. Towers cost money to place, depending on their type, while enemies generate varying amounts of funds for the user depending on their difficulty. The user must combat the drones with each wave generating a stronger group of drones, while managing their money to ensure they do not run out and become unable to defend themselves.

III. <u>Game Instructions</u>:

The game was created specifically to run on the Google Chrome web browser. The game was built using the WebGL platform within Unity. Because the game was designed for the WebGL platform the game can be accessed using other web browsers that support HTML5. For example, the game can also be accessed using Apple's Safari web browser.

To access the game, first open Google Chrome. Then type in the web address below:
http://people.oregonstate.edu/~has/NoDroneZone/

If the page loads and an error specifying that the web browser does not support WebGL or the game isn't loading, follow the instructions to install the extension GameLoad for Google Chrome. The web address for game load is below:
http://www.gameload.top/

When the game loads, the user will be brought to the Login and Registration screen. Prior to getting access to the game, the user needs to have an account that is stored on the server. To register, the user will fill out a form that requires their name, email address, and password. Only one account is allowed per Email Address.

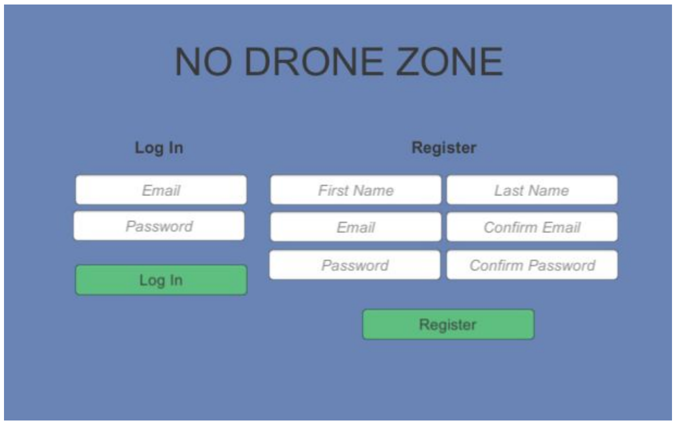
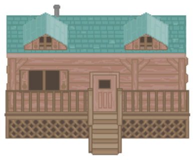


Figure 1: Login & Registration Screen

The game will load when the user successfully registers an account or logs in using previously registered credentials.

The game contains 3 levels which the user can advance through linearly. The user will need to defend their home for each level. The user has 3 lives total for the duration of the game, and life is worth 5 hitpoints. Each level generates 10 waves of enemy drones. Each drone that makes it to the user's home will reduce their HP by 1 hitpoint. Thus, if 15 enemy drones make it to the user's home during the duration of the game, the user will have lost the game.

The targets of the enemy drones for the various levels are shown below.

Level 1	
Level 2	
Level 3	

GUI Layout and Description:

Wave Indicator	Message that will appear when the game is going to generate a new wave of enemy drones. Message appears when the game is within 3 seconds of generating a new wave.
Level Information	Displays the level and wave information. The wave value displays the current number of waves the game has generated for the level.
Bank	The bank shows the user how much money they to spend on towers. This value changed real-time as the user spends money on towers and earns money by destroying drones.
Towers	The towers section displays the towers that the user is able to purchase, and the cost of each tower. If the user does not have enough money in the bank to purchase a particular tower, the tower will be grayed out.
User Health	Shows the number of lives and the current number of hitpoints remaining in that life.

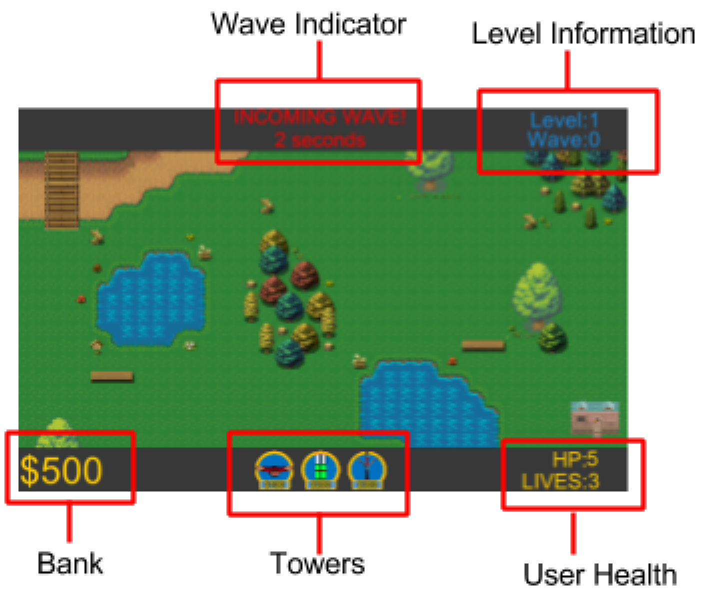
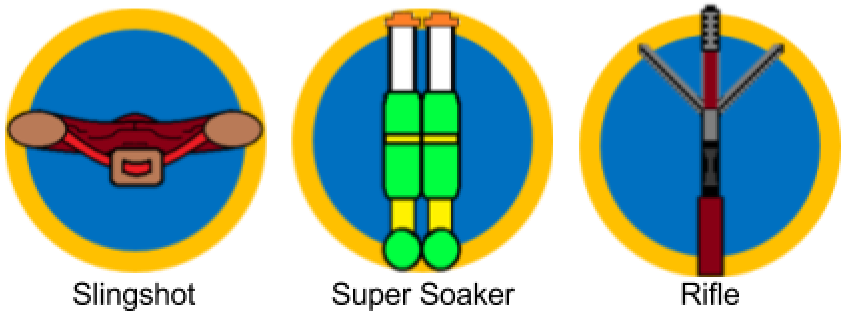


Figure 2: GUI Layout

Placing Towers:

There are three distinct tower types: Slingshot, Super Soaker, and Rifle.



The sprites for these were created by a member of the team in MS Paint and Powerpoint. Each tower has a set cost. The player must have sufficient funds in order to be able to select a tower for placement. Each tower has unique values for its range, rate of fire, projectile speed, and damage. Towers will automatically target and fire on enemies that enter their range. They will continue firing on an enemy until it leaves their range. The tower will then target the enemy in their range which entered it first. To place a tower, hover the mouse over the desired tower and left click with the mouse.

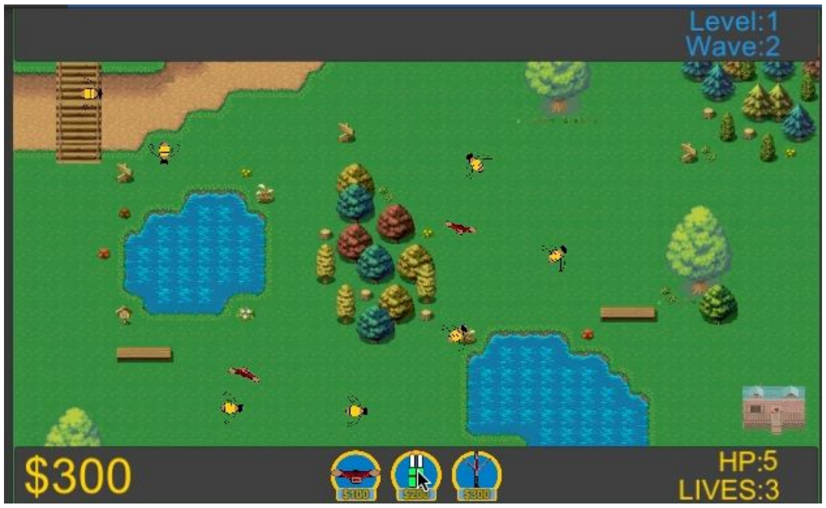


Figure 3: Select Tower

Move the mouse to the desired placement spot. Spots where the tower can be placed will turn green, indicating the tower can be placed there. If the location does not turn green, a tower cannot be placed there. Spots which can hold a tower but already have one placed there will turn red. To place the tower, simply click with the left mouse button on an acceptable location. To cancel tower placement, hit the spacebar.

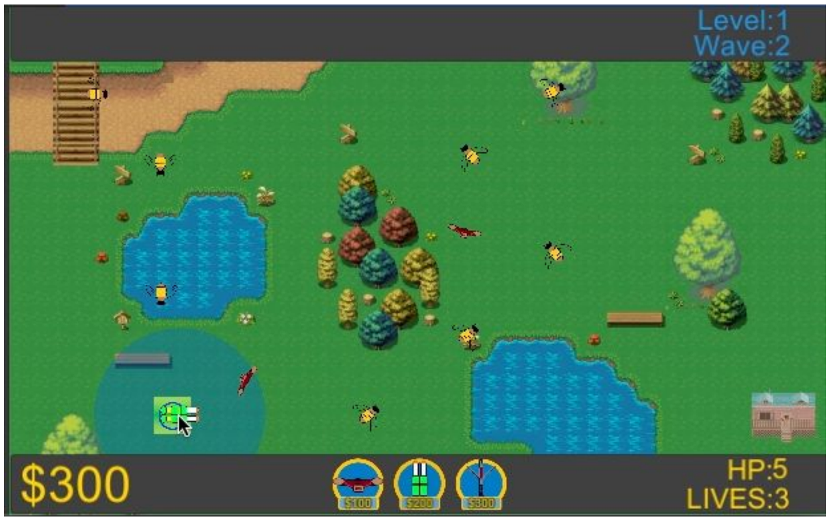


Figure 4: Available Spot for Tower Placement

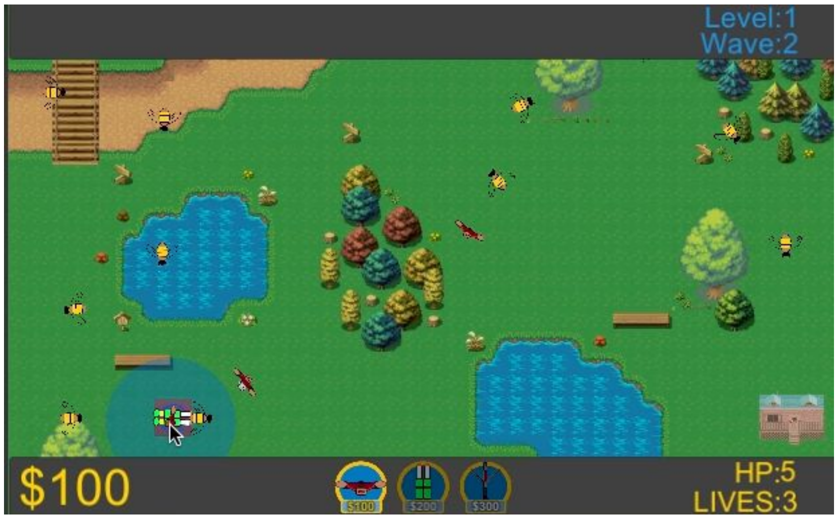
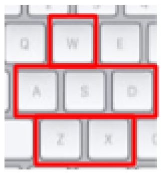


Figure 5: Unavailable Sport for Tower Placement

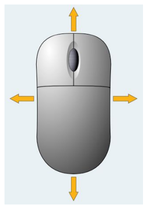
Game Controls:
At the start of the game, the game is zoomed out. If the user zooms in during the game, they can pan around the map. Moving the mouse cursor close to the game's window will also allow the user to pan around the map. For example, moving the cursor to the top edge of the window will pan the game up if the user is zoomed in.

Keyboard:

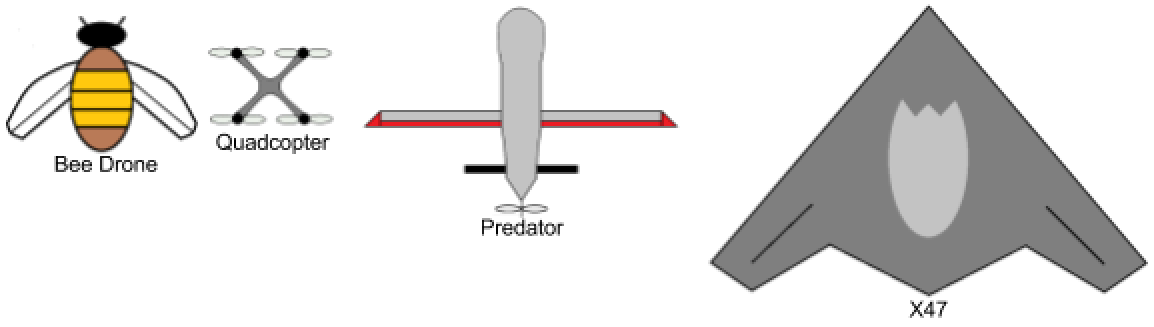
	W: Pan up S: Pan down A: Pan left D: Pan right Z: Zoom out X: Zoom in

	Spacebar: Cancel tower placement and/or tower range display
	Esc: Pauses game and displays Save/Quit Menu. Pressing Esc again will unpause the game.

Mouse:

	Left Click: Select Tower/ Place Tower
	Scroll: Zoom In and Zoom Out
	Mouse Movement: Pan Around Level if Zoomed In

Enemy Spawning:
Once the player has logged in, a countdown will begin at the top of the screen. Once the countdown reaches 0, enemies will begin spawning. There are four enemy types: Bee, Quadcopter, Predator, and X47.



Like the tower sprites, the enemy sprites were created by a team member in MS Paint and Powerpoint. Each enemy has unique stats for health, speed, and money generated upon their destruction. Enemies will proceed on a set path to their target. If an enemy reaches the target, the player will lose one HP. If the player's HP reaches 0, they will lose one life. If their lives reach 0, the game will be over.

Pause Menu:
Pressing Esc while playing the game will bring up the Save/Quit menu.

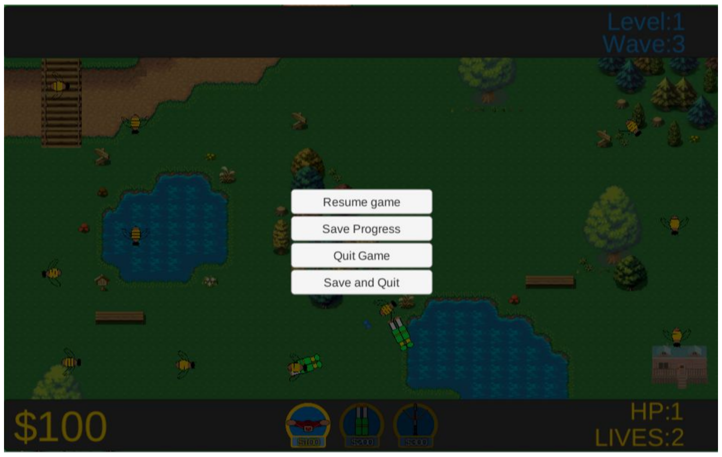


Figure 6: Pause Menu

Selecting 'Resume Game' will return the user to the game.
Selecting 'Save Progress' will save the user stats the user had at the start of a level. Note that the game will not save unless the user has completed at least the first level, to prevent getting "stuck".
Selecting 'Quit Game' will end the game and return the user to the Login & Registration screen.
Selecting 'Save and Quit" will save the game and then end the game for the user, at which point the user will be returned to the Login & Registration screen.

Game Over Menu:
The user will be brought to the game over screen when the game ends. The game ends when the user has defeated each level or if the user's hitpoints and lives have reached 0. The "Score" represents the amount of money the user had when the game ended. So a perfectionist can strive to complete the game with as much money remaining as possible.

The user will have the option to replay or quit the game.
Selecting 'Replay' will reset the user's stats and restart the user at level 1.
Selecting 'Quit' will return the user to the Login & Registration screen.

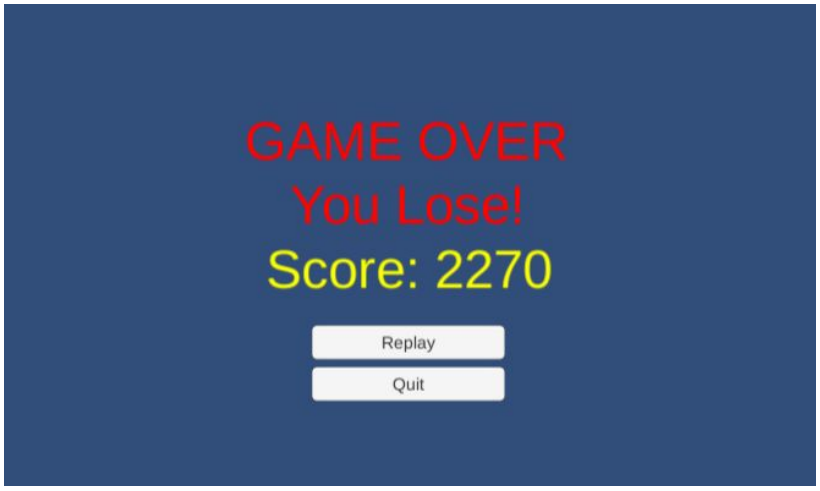

GAME OVER
You Lose!
Score: 2270

Figure 7: Game Over Lose Screen

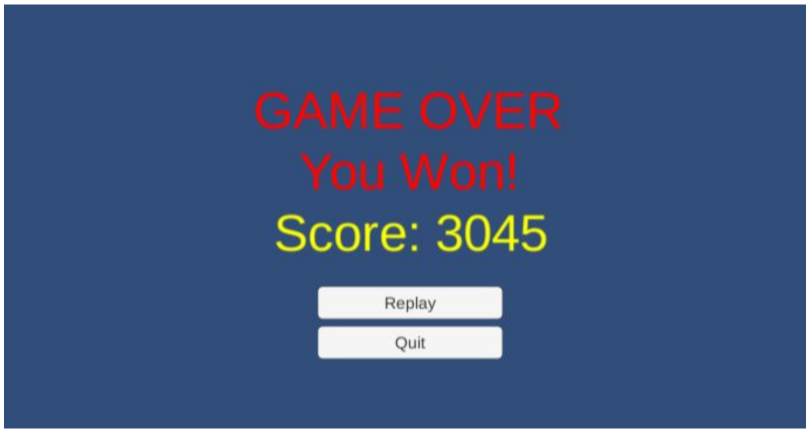

GAME OVER
You Won!
Score: 3045

Figure 8: Game Over Win Screen

NOTE: The next 3 figures show the path of the enemy for each level. Do not look at the figures if you want to learn the path by playing the game.

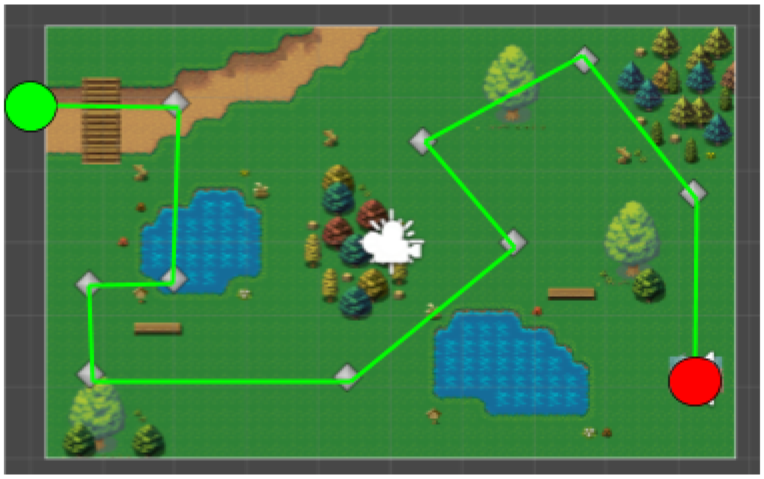


Figure 9: Enemy Path for Park Scene

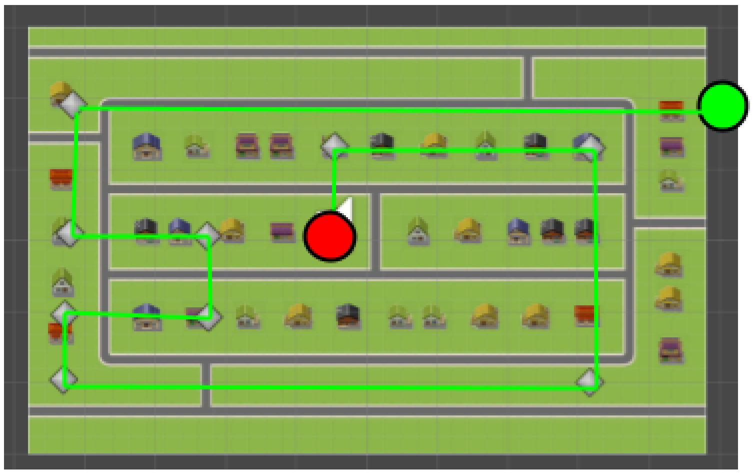


Figure 10: Enemy Path for Neighborhood Scene

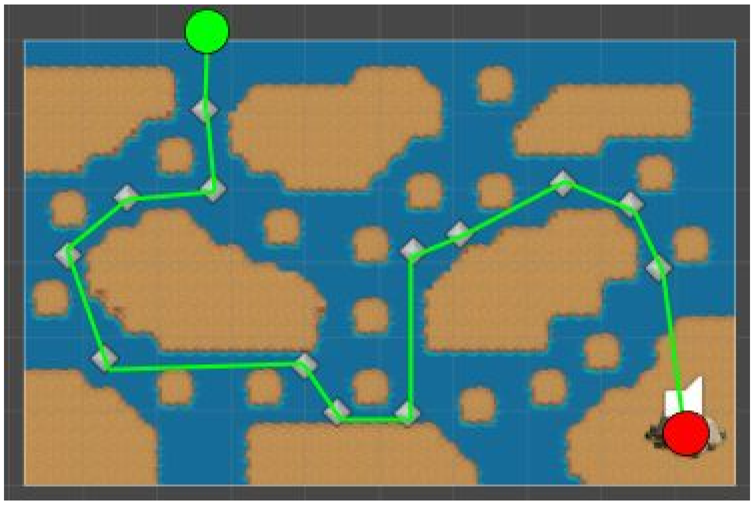


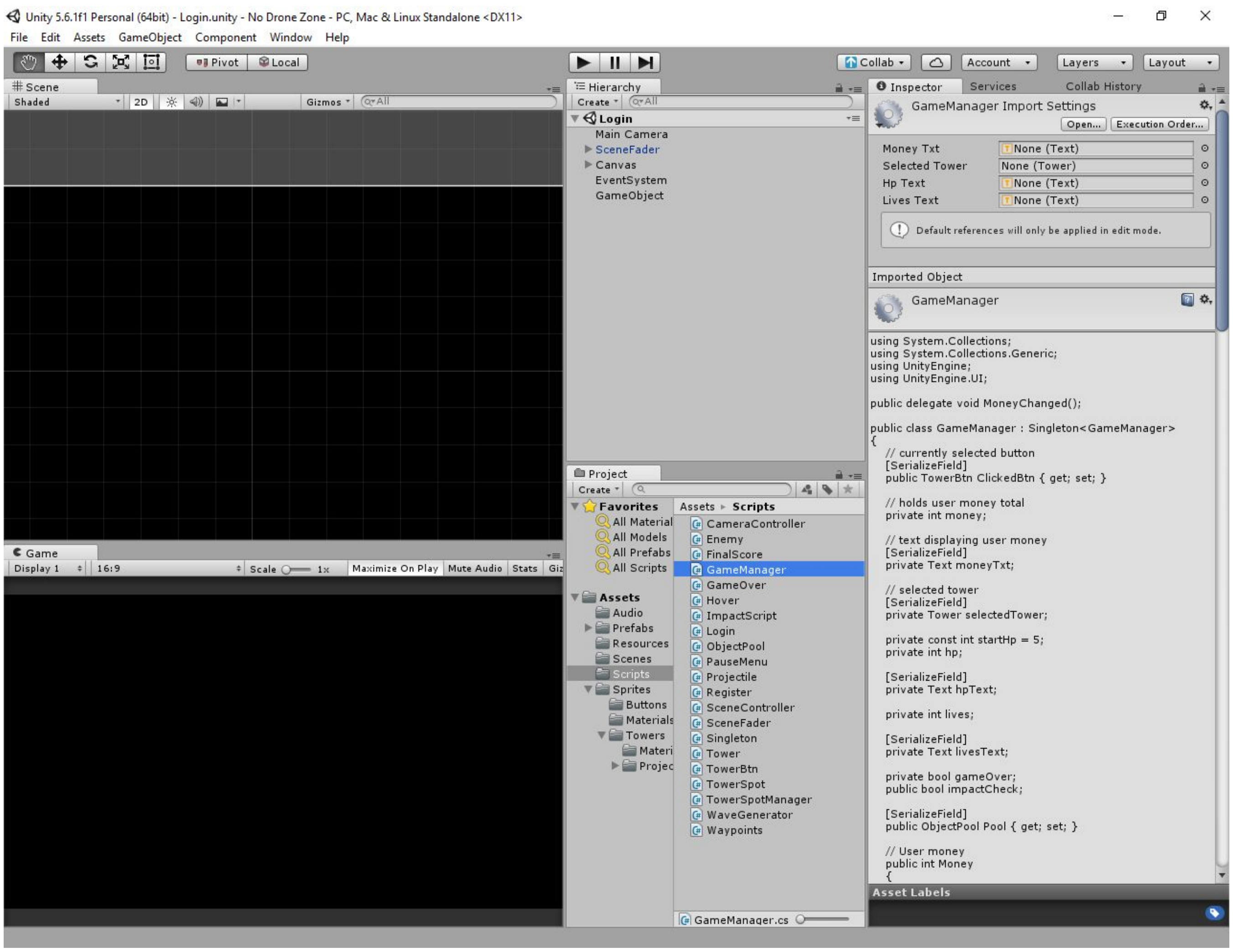
Figure 11: Enemy Path for World Scene

IV. Software and System Functions:

A. Unity Game Engine

For the gameplay code, the team utilized the Unity Game Engine. Unity is an excellent engine with a diverse selection of tools which can be used to develop a variety of game types. It can be downloaded for free from https://unity3d.com/. Unity requires you to create a free account in order to user it. Once an account has been created, the No Drone Zone project can be opened, and the code and assets can be viewed.

Display:



Money Txt	None (Text)
Selected Tower	None (Tower)
Hp Text	None (Text)
Lives Text	None (Text)

Default references will only be applied in edit mode.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public delegate void MoneyChanged();

public class GameManager : Singleton<GameManager>
{
    // currently selected button
    [SerializeField]
    public TowerBtn ClickedBtn { get; set; }

    // holds user money total
    private int money;

    // text displaying user money
    [SerializeField]
    private Text moneyTxt;

    // selected tower
    [SerializeField]
    private Tower selectedTower;

    private const int startHp = 5;
    private int hp;

    [SerializeField]
    private Text hpText;

    private int lives;

    [SerializeField]
    private Text livesText;

    private bool gameOver;
    public bool impactCheck;

    [SerializeField]
    public ObjectPool Pool { get; set; }

    // User money
    public int Money
    {
```

The main Unity display provides an easy to use layout. From here the develop can add and modify game objects, create scenes, import resources and run the game. This is ideal for quick testing. Additionally, scripts can be created here and then launched in Visual Studio which is integrated with Unity.

Assets:
Unity utilizes a variety of assets to generate the game.

a. Scripts
   Unity utilizes C# as the language for game scripts. These scripts are used to create classes which define the behaviour of game object. In our project, we had scripts governing everything from the enemies and towers, to the scene controller which handled transitioning between the login screen and the various levels. Scripts handled the updating of all on screen text, and a variety of other game components.

b. Prefabs
   A Prefabs is a reusable game object with defined behaviour and properties attached to it. When a prefab is placed in a scene, a unique instance of it is created. This instance is

linked to the original prefab. We created prefabs for our towers, enemies, tower buttons, and projectiles. These enabled us to govern the behaviour of the objects, while still providing the flexibility to assign different properties to different types of objects.

c. Resources
For our project, the resources consisted of text files which contained information defining the properties of the towers, enemies, the placement of tower spots, and the generation of enemy waves. These files allowed us to easily modify the properties during testing and were used extensively in the game balance phase of our project.

d. Scenes
Scenes contain the objects of the game. They can be used to create the game menu, levels, and a variety of other things. We utilized scenes for the login screen, park, neighborhood, and world levels, and the game over screen. All our game objects were placed in the various scenes to generate the actual game.

e. Sprites
Sprites are the graphic objects which are used to represent the various game objects in the game on the screen. For our project, we elected to create our map, tower, tower spot, tower buttons, projectile, and enemy sprites. These were used by the SpriteRenderer component of game objects to display the graphics. Animations were created using sprites to display enemies moving, towers firing, and enemies reaching their target.

B. Visual Studio
All the scripts for the game were written in Visual Studio. This IDE is tied to Unity for script development and debugging. It was a key tool in the development of game behaviour, and was used extensively throughout the project. Scripts were created in Unity and then auto populated with basic class layouts for ease of development. The debugging feature in Visual Studio can be attached to Unity to allow developers to step through the code to isolate bugs while the game is running.

V. Tools:

A. Unity

Unity was the main software that was used between the group members. Aside from being the workhorse behind the ability to create the game itself, Unity's version control system allowed the group to be up to date. This proved to be very convenient because it meant that the team had one less system to worry about. As soon as anyone of the members open up Unity, they instantly knew if their files were up to date or not.

B. Tiled

The software Tiled was used to create the background images used for each level in the game. The desire of the team was to create custom backgrounds for each level. The tilesets were sourced from the internet. Tiled allowed us to combine the different elements from each tileset to create the backgrounds that would be used in the game. One of the best things about Tiled was that it allowed us to place the tiles at various layers. This made it easy to fix mistakes and make changes without having to redo large sections of the maps.

C. MySQL/PHP

An important feature of the game was allowing the user to log in and start off at a previously saved level. This meant that the game needed access to a database and the ability to retrieve and push from that database. The OSU ONID Database Server was used to hold the data that the game would need to function. PHP scripts were created for three different cases.

1) A user logged in and the game needed to load the user's data that is stored in the database.
2) A user registered and the database needed to store a new account in the database.
3) A user wanted to save their current state and the database needed to be updated.

D. MS Paint

A key tool in any graphic artists arsenal, MS Paint was crucial in the development of our high quality sprites. Each sprite was lovingly created by hand, using the state of the art tools which MS Paint provided.

E. Powerpoint

Another highly advanced tool, Microsoft Powerpoint was utilized to refine the sprites created in MS Paint. This included creating simple shapes, as well as removing backgrounds.

VI. Team Member Accomplishments:

A. Steven Ha

Steven implemented the controller that would be used to load the scenes that made up the game. This included creating a transition script that would fade out of one scene and fade into the next scene. He was also in charge of implementing the process that would allow the user to register and log into the game. This included creating the database tables and accompanying PHP scripts that would be used to save the user's game stats in the event that they needed to end the game early. Steven created the controllers that would read in the path the enemy units would take for each map, as well as creating the controller that would generate the enemy units for each map by reading in the appropriate enemy wave data from text files. Steven also designed the background images that would be used for each level of the game.

Steven also served as the unofficial leader of the group making sure that everyone was completing their weekly tasks on time. He was also in charge of setting up the meetings to discuss issues, etc that everyone was facing.

B. Thomas Mathew

Tom's focus was the drones, including design, creation, instantiation, movement, animation, interaction with tower projectiles, and user feedback. The design process went as follows: hand sketch mockups (4 out of 11 ideas were picked), draw the drones in MS Paint, use Powerpoint to remove whitespace, and finally add to Unity as Game Asset Sprites. Next was movement through the level's waypoints correctly to the end. Attempted use of animations at each waypoint was abandoned in place of the more convenient "Quaternion", "AngleAxis", and "transform.rotation". The next step was to give drones "Health", cause them to take damage when fired upon by towers, and destroying after enough damage is taken. Finally, two sound effects were implemented when a drone is "shot down" by the towers, and when a drone is able to cause damage, as well as an explosion animation for the latter. Tom also developed a Pause Menu for the game with four buttons: Resume, Save, Quit, Save and Quit. Saving was tricky, as there was some confusion with variable names and being unable to save to the database. Finally, Tom worked on playtesting and debugging, finding problems with returning users having the wrong amount of money to start the game, and with GameOver->retry causing a similar problem. Steve was able to track down the former and Tom was able to fix the latter via towerrefund() function.

C. Gabriel Orellana

Gabriel was responsible for the development of towers, to include ranges, projectiles, targeting and firing on enemies. This included creating the sprites for the towers, tower buttons, tower spots, and projectiles. He developed the scripts handling the placement of towers, including determining valid locations via the creation and placement of tower spots. He developed the tower scripts to handle the generation of towers, including loading their properties from files, targeting enemies, and firing on them. Additionally, he modified the loading properties code for use with enemies and their properties. He developed the scripts for projectiles to handle their tracking and damaging enemies. Further, he created the Game Manager script which handled the tracking of user money, HP, and lives, as well as the conditions for game over. He also assisted in the integration and testing for bugs in the game. Finally, he was responsible for handling game balance, ensuring the game was not too easy or difficult.

VII. <u>Deviation from the Original Project Plan</u>:

Generally speaking, the group was able to stick to the desired structure of the game that was defined at the start of this class. A few of the features that we weren't able to implement due to scheduling issues include the ability to keep track of the high scores. Besides beating the game, we wanted a feature that would make the user want to play the game again and showing the high scores would have added an additional layer of competition to the game.

Another item that changed the from the original project plan was the creation of a 'Shop Menu' where the user could buy various towers. The original project plan specified 7 tower concepts. The team ultimately decided on 3 towers for the game. The reduction in number of towers, made the Shop Menu idea less appealing. Also due to the game only having 3 levels, the change was made to streamline the tower purchasing process and give the user the ability to purchase a tower during the gameplay given they have sufficient funds.

The group also wanted additional sound effects for the game. This included background music, tower firing sound effect, and drone taking damage effect. Due to time, these sound effects were removed from the plan with the only sound effects included in the game were a "splat" sound when a drone is destroyed and an explosion sound effect when a drone makes it to the level's home. Removing a majority of the sound effects from the game also removed the plan of having an 'Audio Menu' that would allow the user to specify the volume for the various sound effects.

Also removed due to time was including the instructions in the game so the user would know how to play the game. We felt that if we had more time to work on the game, including the instructions in the game would be the most beneficial feature to the user.

Our individual schedules also deviated after the first week of implementing the game. Our schedules put a good amount of emphasis on game graphics at the start of the project, this included the background, enemy drones, and towers. After the first week of working on the various sprites and images, we knew that we needed to focus more on the features of the game. It should also be noted that due to the nature of the project, team members were also helping each other complete their tasks. We knew that this would be the case prior to starting the project, but didn't expect it to be the norm near the end of the project.

VIII. Conclusion:

Working on this project was a challenging yet fun learning experience. From a technical standpoint, none of us had used the Unity tool before, and we had little or no experience coding in C#. We (correctly) allocated a significant amount of time for learning, which we used in walking through the official Unity tutorials, reading Unity documentation, reading about C# on stackoverflow, and more. Still, there were a number of instances where a lack of knowledge led to wasted time, often involving the interaction between Unity, game Objects, prefabs, and the C# scripts. For example, Tom wasted a number of hours trying to implement rotation animations at each waypoint before discovering a few functions that largely handle and automate movement and rotation. We realized relatively early on that the best way to implement a playable and fun game is to mix hard-coding with flexible code. For example, tower placement locations and waypoints for each level were fixed so that the user would be able to understand what was and would happen, and could learn from playing. Whereas coding the movement of the drones in a flexible way saved time, as opposed to trying to hard-code behavior at each waypoint.

We also learned a number of lessons in teamwork. Steven did a great job stepping up to the plate and driving forward the process, holding everyone accountable for completing tasks on time. It can be easy to lose momentum over an 10 week span and having a de facto leader kept us moving. As a team, we also did a good job of being willing to help out as needed on the project and tackle tasks that were unassigned/unanticipated, or if someone else was stuck on a feature. A few examples include Gabe helping Tom to understand how certain Game Objects and Scripts in the game were working together, Tom helping Gabe fix a problem with towers targeting the drones, and Steven improving Tom’s pause menu and eliminating scene transition delays when Tom got stuck. Well-commented code and constant communication (not to mention the easy collaboration tool in Unity) were key to understanding and building upon each other's work.

Finally, we learned that a “close” to finished game isn’t necessarily so. Even though a game might be playable on a basic level by the developers who know how the game is “supposed” to be played, there can still be many issues to fix before it’s ready to be played by users. Whether it was a bug that allowed users to re-start and gain money for free (potentially infinitely), delays in scene loading, or game balance (the game was much too difficult at first), games (and by extension, any software to be used by others) require significant fine-tuning and testing before “launch”.

We enjoyed putting this game together and wish you luck in getting rid of any pesky drones you see!