Defend the Tower Report



Game Programming and System Architecture

Keiran Millar | Computer Games Technology | 1502338

# Introduction

The aim of this project was to create an application which utilises Augmented Reality (AR). Augmented Reality is the process of using combining the real world with the virtual world in real time. Such as this game which will display a game over the top of camera feed from the PSVita. The game uses markers to place a tower which you must defend. To defend the tower, you can stand on the enemies to kill them or you can place traps using the other markers in the real world. Each round gives you a new “trap” to kill the enemies with, this can be repositioned by moving the other PSVita markers (2 and 3).

# Augmented Reality

Augmented reality has been available in games for many years. Augmented Reality can be seen in use with the EyeToy for PlayStation 2 all the way back in 2003. EyeToy: Play released February 2003 and allowed players to choose from twelve mini-games. Most of these games used the camera feed and would allow you to hit incoming projectiles (footballs, CDs, ninjas etc.). Another form of Augmented Reality is using location-based services, most commonly used on mobile phones but can be utilised in the Nintendo 3DS and PSVita. One of the biggest games of 2016 was Pokemon Go which utilised these location-based services to show a map of your area, where you could then physically go out and walk to find more Pokemon. These Pokemon could then be used to battle other players for superiority over “gyms”. The game was a massive success and according to App Annie, generated $950 million in the year of 2016.

AR also has other uses than games, for example, IKEA utilises AR in their app on mobile to allow you to place furniture from their stores into your house. This allows a customer to see if the furniture fits and whether it matches the rest of the room or not. Augmented Reality is also used for 3D model viewing, this can prove invaluable for designer as it allows them to see their creation from any angle rather than using a mouse and keyboard/graphics table to spin a camera around their project. Microsoft have furthered this technology with their HoloLens, this headset allows a user to overlay various elements over what they can see through the headset. In a promotional video for the HoloLens, Microsoft shows off the HoloLens’ ability to display models in front of the user and allow them to physically walk around and investigate the model. In a different section of the promotional video, the HoloLens is shown to be able to make Skype calls as the user is walking around the office and it can even play games using the room around the player as the play space. A downside to the HoloLens however is that the window where graphics are overlaid on the real world is reported to be fairly small and restrictive, giving a feeling of tunnel vision and ultimately can break the immersion for the player since their whole peripheral vision isn’t filled with the digital world.

This project has utilised AR markers to function as the reference points for the application, marker 1 is used as the main reference point and subsequent markers are used for the different traps in the game.

# Application Design

Five classes were used in this project, the main ar\_app class, a timer class, menu class and both player and enemy classes. These were used in conjunction with a state machine in the ar\_app class to play the game. The menu class was designed to update and render all of the menu screens and allow transition between menus using the state machine. The timer class is used to start a timer and show how long the player has survived. The player class is used to manipulate the player, through this class the player can move around the world and can have their health reduced when enemies attack their tower. The enemy class has functions which allow for them to move towards the objective (is the tower but can easily be changed to another MeshInstance). The enemies can also be killed when they hit the tower or are stood on by the player. Stored within the ar\_app class is the main state machine which allows transition between, splash screen, instructions screen, main game, pause screen, next round screen and finally the game over screen. Ar\_app also contains the handling of user inputs, marker recognition, collision detection as well as the main games update and render functions. Inside of ar\_app is where the calls are made to the other classes, for example, this is where the game calls the enemy class to update their positions every frame.

For the gameplay, I decided to limit the user to only two traps as having more allowed for the player to simply create a wall around their tower, effectively ruining the game as no enemies would be able to get past the barrier. I also decided on a timer between rounds rather than a button prompt to start the next round as having a timer puts more pressure on the player to quickly place down their defences for the incoming round. For playing the game I also made a conscious effort to keep all the gameplay controls on the left side of the PSVita (except the pause button) as this allows the players right hand to be used to move cards around on the desk, should the need arise.

Shown below are two diagrams, one is the flow chart showing all of the possible states it can be in, and how to transition between each of these states. The other is a Unified Modelling Language diagram which shows the important functions and variables in each of the classes used in my project. It also shows how each of these classes interact with each other, by using arrows to show where information is sent and received.

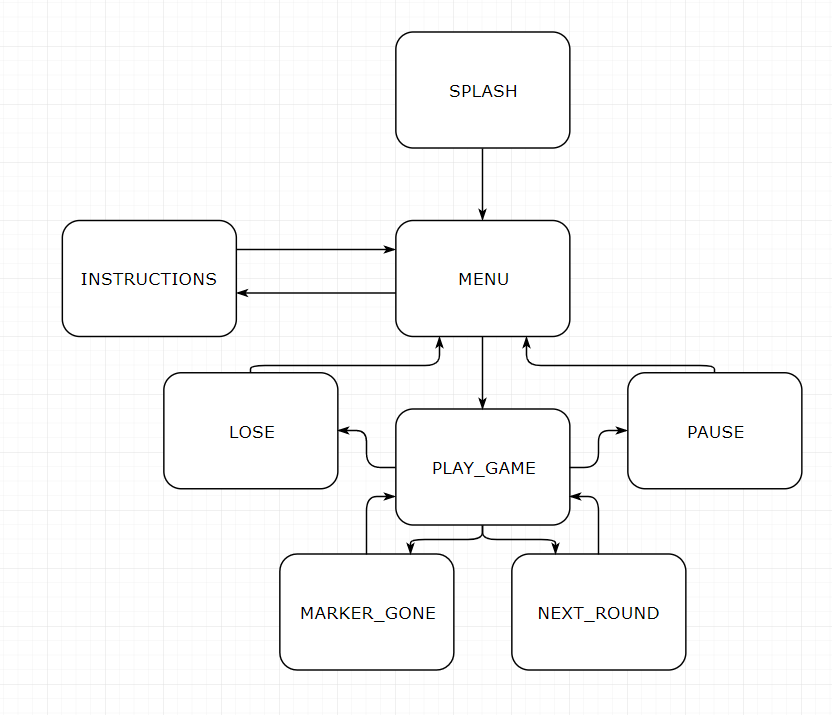


Figure 1: State Machine

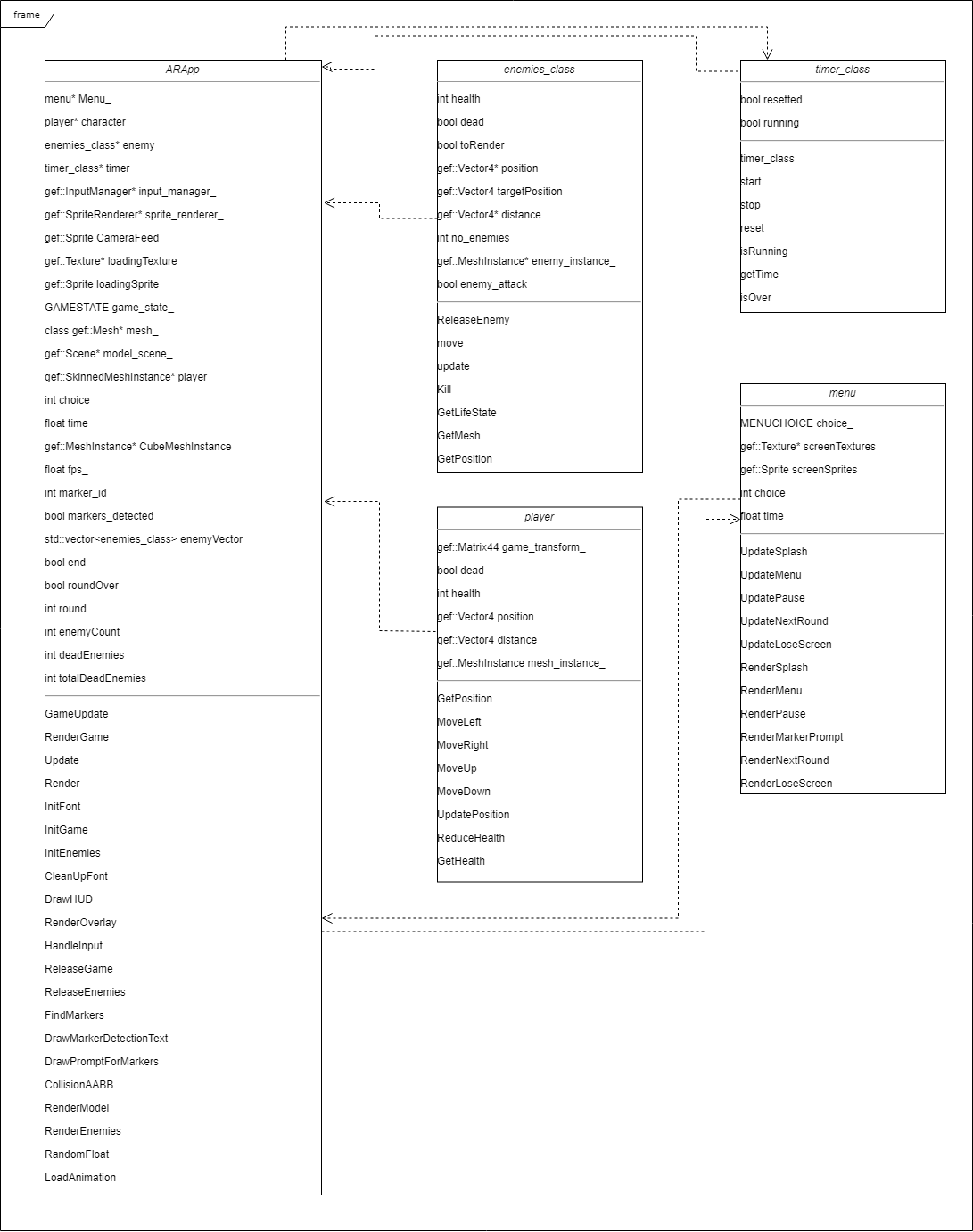


Figure 2: UML Diagram

# Techniques Used

Inside of the main ar\_app file is the main gameplay functions and the state machine for the whole application. The update and render functions are used to decide what should be done based on the state machine’s current variable. I will cover the menu functionality later on when I discuss the menu class itself but for now I will focus on the game itself. When the game is started all of the required variables and models are loaded in, the round and enemy counters are reset, and the enemies are initialised. To initialise the enemies a vector is used to hold each instance of the enemy class, a vector was chosen as since the size of the vector can be added to, it allows there to be more enemies as the rounds go on. The game then attempts to find the markers, if none of the markers are seen then the game will transition to the marker lost state. If marker 1 is found, then the game will begin, and the tower will be rendered on top of the marker transform and the game will be rendered using this transform as the origin. Every loop of the game update function will then move the enemies closer to the tower and will check whether the player has hit the enemies or if the enemies have hit the tower. Using AABB collision detection, if these collisions happen then the enemy is killed, and the kill counter is incremented. There are two kill counters, one is the total kills, and this is used purely to show the player how many kills they have. The other is a counter of how many kills have happened in this current round, when the current rounds kills are equal to the number of enemies in the round, the round is ended, and the enemy vector is deleted. When a new round starts, the enemy vector is initialised to be the size of the required enemy count for that round. When the enemies attack the tower enough to kill the player, the enemies are deleted and the game is taken back to the main menu after releasing the model and animations of the player.

Within the menu class are render and update functions for each of the menus in the game. The splash screen, next round screen and game lose screen are all finished after a set amount of time, so they add up the frame time from the main application until the timer has passed a set threshold (splash and next round being two seconds, game lose screen being 5 seconds to allow the player to read their score). The instructions screen update function simply waits until the user presses circle and then takes them back to the main menu. The menu and pause update functions return a float based on the selected option in the menu as seen below:

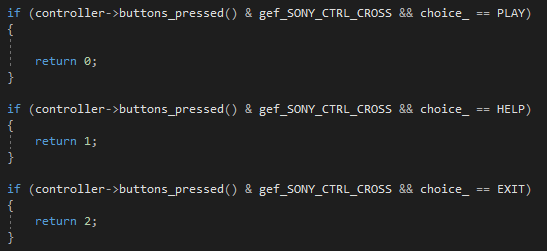


Figure 3: Segment of the Menu Update Function

Based on which number this function returns, the state machine can act accordingly to transition the user into their desired game state. This function is also seen in the pause update function but only has two possible number it can return as the pause menu only has resume and quit, although it would be very easy to implement more menu options should the need arise.

When the menu class is initialised at the start of the application, all of the textures for the menus are loaded in and made into sprites, rendering these sprites are then as simple as just drawing them to the screen using the sprite renderer. For the main menu and the pause screen, a marker select icon is used to display which option the user is currently selecting, pressing up and down on the d-pad moves the selection up and down, resulting in the select sprite also being moved. For the render marker prompt function, firstly I render a blank sprite with no text, then I render the marker sprite. This tells the player to point the camera towards marker 1 if it has been lost from sight

The enemy class is used to hold the location of the enemies after they are spawned into the game. Using their location and the location of the target they want to attack, I calculate a vector that is the distance between the enemy and their target. To then move the enemy, I divide this number by 15 (other numbers can be used, a higher number results in a slower enemy) and it the vector to their position, resulting in the enemy making progress to their target every time the move function is called. The enemies also have a variable called dead, which when it is true allows for me to not render them or calculate their movement in the game, this functionality allows the game to not waste time calculating movement for enemies who are already dead.

The player class is used to move the player through the game world by adding or subtracting from their position based on the direction the left stick is moved, pressing the left bumper button also allows the player to run. Inside the player class is also the players health for the game and a function which can be called to reduce the health when enemies hit the tower.

# User Guides

This game requires a flat surface to be played on, when starting the game place, the first marker in the middle of the play space and then start the game by pressing play while holding the PSVita in a position where the rear camera can see said marker. This marker is the reference point for game world so moving this will move the “game world”. From here your tower will be spawned on top of the marker and enemies will start to go towards your tower. Using the left analogue stick and the Left bumper button to sprint, you must stand on all of the enemies to kill them. If an enemy reaches your tower, then they will reduce your health and will be killed. If enough enemies attack your tower, you will lose the game. After your tower is destroyed, it will display how many kills you have had in that game. Pressing the pause button during gameplay will pause the game and can be resumed at any time by simply pressing resume or pressing the pause button again. After the game is over, players will be brought back to the main menu where they can play another game or quit the application.

After all of the enemies in a round are killed, the game will go to a screen warning you of another round, round 2 will unlock a trap which can be placed using the second marker available to the PSVita. Placing the traps allows for the player to kill enemies coming from a certain direction as when an enemy makes contact with the trap, they die. You can have two total traps (the second is unlocked in round 3) which allows the player to help fight off the ever-increasing number of enemies.

# Critical Analysis

To improve my application, there are many aspects which could be tweaked to make the users have a more enjoyable experience. One of the biggest things I would have liked to have gotten working was displaying the camera feed behind the prompt when marker 1 was lost. I attempted to implement this feature as you can see by the fact my marker\_screen texture actually has a transparent background which I had intended to allow the user to see the camera feed behind them but no matter what I did, I just couldn’t get the render function to work as intended, usually resulting in a cornflour blue background and the marker\_screen sprite being rendered on top.

The game could be improved vastly by using models and animations for the enemies and the tower/ traps, instead of using the default cubes. However, during the creation of this game I focused more on the mechanics and put the models to the bottom of the priority list and ultimately, I ran out of time before implementing any other models than the mech used for the player.

Another improvement I would like to implement is when the round starts, I could save the location of the trap markers with respect to the tower marker. This would be done by multiplying the traps marker transform by the inverse of the origins transform. Doing this would allow me to save where the traps where placed meaning that the player wouldn’t be able to move the traps after the round starts. Not only would this stop the player moving the traps, it would also allow me to render the traps without the need for the traps markers to be present and rely on the origin marker for all of the rendering. This would hopefully reduce the amount of time the traps would be lost due to not seeing the markers as only the one marker would be required for the play the game.

To help further clean up the ar\_app file, I would like to have given the game its own class so that there would be less functions inside the ar\_app.cpp, at first I thought the program would be fine if I was to give the menus there own class and leave the “main” gameplay in the ar\_app but obviously as development continued the functions required grew and grew to what it is now. So, using this experience, if I was to create this project again, I would be able to make a more effective judgment on how my classes should be laid out and how they will interact with each other.

# Conclusion

Augmented Reality does have its limitations, but when a game is designed around these limitations, the end result can be quite immersive. One of the biggest decisions that should be made early on is how the game will find its origin point. Some games could benefit more from scene mapping and using the environment around the player as an origin, for example a game where you have to look around the room or move the camera around a lot may be more suited to that approach of finding an origin point. This is due to how finicky it can be to recognise a marker when the camera isn’t being held at an optimal angle, the smallest of obstructions in front of the markers can stop them from being recognised. For example, when creating my game, sometimes the exposure of the camera would be too exposed, causing the camera to just not be able to find any markers. Another problem I faced was that if you moved the camera too quickly, the camera wouldn’t be able to detect the marker as the image would be blurred, using the vita itself as a reference point in the world would have helped to eliminate this issue.

Another important aspect of design that I didn’t think of straight away was considering where the players hands were during gameplay. When creating a normal game, it is easy to forget this as it usually isn’t important to the overall design of the game but when creating this AR project, I had to be aware of whether the player had a free hand to move the markers on the desk in front of them. Due to this, I made sure that all of the gameplay controls where on one side on the vita to ensure that the player had a hand free to move their defences if they needed to. Although, it is still fairly awkward to hold the vita and control the game all from the one hand. Using a headset to view the game world (such as HoloLens) would allow for a much more immersive experience and allows for the player to use their hands for purely gameplay controls.

# References

PlayStation (2018): EyeToy: Play

Available at: <https://www.playstation.com/en-gb/games/eyetoy-play-ps2/>

Venture Beat: Pokémon Go generated revenues of $950 million in 2016

Available at: <https://venturebeat.com/2017/01/17/pokemon-go-generated-revenues-of-950-million-in-2016/>

Ikea: IKEA Apps

Available at: <https://www.ikea.com/gb/en/customer-service/ikea-apps/>

Doug T: Random Float function

Available at: <https://stackoverflow.com/questions/5289613/generate-random-float-between-two-floats>

Microsoft: HoloLens

Available at: https://www.microsoft.com/en-us/hololens