CMP404 Applied Games Technology

**Introduction:**

As part of the CMP 404 module, a game application was to be developed that made use of Augmented Reality Technology using the PlayStation Vita. Having had previous experience playing tower defence games, it was decided that a game in this style would be ideal for this project as it would create multiple opportunities to showcase Augmented Reality Technology as a core gameplay mechanic.

**Instructions:**

* Game Begins in Splash Screen, press any button to continue to Menu
* In Menu State, press “x” on the Vita to launch into the Game State from play button.
* Player can also choose to close the game or the Help Screen
* In Game State, Marker 1 is needed to draw map
* The Game starts with Towers on Mark 2 & 3 for the player to begin with, these can be moved to players desire before start of the round, once round starts only the cannons can be moved
* Round Starts by hitting play button in GUI Window and Enemies will spawn and move through map
* If too many Enemies reach the end, Player loses
* Player can target enemies with crosshair in centre of screen and press “o” to fire
* If all Enemies are killed, round ends and player can now use the next marker to build a new tower or upgrade an existing one by tapping the tower and choosing an option from GUI window
* To start next round, hit play button again
* Survive three rounds to win game
* Press “x” to return to Splash Screen from any non-Game State screen

Menu Screen:

A screenshot of a video game

Description automatically generated

**Research & Design:**

AR as an Industry:

Whilst AR can feel like a trendy new gimmick for game design, due to the flexibility in its delivery and implementation, it has a much wider reaching potential across global industries. The main requirements are a camera and a display, these are both basic components of every smartphone in world and with an estimated 3.8 billion smartphone users (Statistica, 2019), there is a large opportunity for development within this area of technology. In fact, research show that the AR market had a value of 11.14 billion in 2018 and is expected to reach 60.55 billion in 2023, forecasting a 40.29% annual growth rate over 5 years (Evangelista, et al., 2020). Proving its flexibility, AR is not restricted to just smartphones. One of the leading areas in AR development is in HMDs (Head Mounted Display). HMDs are a device worn by the user that has a small display unit that can show its user a combination of the real world and computer-generated images and are currently being used by the US Navy for training simulations (Cummings, 2019).

AR in Games:

Whilst development for a smartphone may restrict some aspects of game design, there is still a diverse market with different kinds of games that have achieved some success. The most obvious being *Pokémon GO* (Niantic, 2016), a Geo-Location game that has the player go to specific point in the real world to interact with an aspect of the game world. The success of this game has led to the creation of many clones from different franchises like Jurassic Park and Harry Potter. Another location-based game is The Walking Dead: Our World (Next Games Corporation, 2012), where the player must go to different locations and using the camera must aim at and kill zombies.

Given the experience of playing tower defence games in the past and having identified a number of examples to draw inspiration from, such as Master of Towers (Argaman Creative, 2018), an AR app that uses a Marker system to spawn a towers in game and Castle Must be Mine (The Middle Gray, 2017), a VR Tower defence game available on Steam that showcases the full potential of this genre of game using an emerging technology. The main takeaway from researching these games and going into the development of this project was to try and keep some of the physical real world in the game. As discussed in *Guidelines for Designing Augmented Reality Games* (Wetzel, et al., 2008), two of the key features to a good AR experience are “Do not stay Digital” and “Keep it simple”. In the examples researched, if the game map/level ever filled the screen entirely and none of the real world was displayed, it no longer felt like an AR experience and the game would be better off as a normal game, having the chaos of a Tower Defence game happen on the players desk adds to the fun and immersive nature of the AR games.

Innovations and Moving Forward:

Two of the most exciting areas in technology right now are AR and Ai, so naturally, the way forward for AR would be an integration of machine learning to remove some of its current restrictions, like the need for AR Markers. As discussed in, *Enhancing AR with machine learning* (Rothmann, 2019), AR technology itself is a form of computer vision that compares visuals between camera frames to map and track an environment. By extending that computer vision to the point where the game can recognise what kind of objects the player is looking at, the lines between the game world and the real world can be further blurred and the overall immersion for the player can be enhanced. This type of feature could be very effective for this project specifically, by removing the need for markers and having a machine learning algorithm that can recognise certain object types, like a Mug or a child’s toy, the player now has a more personal investment in the game and is trying to protect their item from the in game enemies instead of an in game asset they have no attachment to. Similar to the work covered in *Human-Centric design personalization of 3D glasses frame in markerless AR* (Huang, et al., 2012), with a sophisticated enough machine learning algorithm for mapping and identifying objects, the game could even render on screen damage to the items, like arrow wounds or scorch marks, as the game progresses. The Lego toy company have recently produced a product called *Hidden Side* (Lego, 2020) that resembles this where the user has a physical toy and using an app the toy is transferred to a game world on screen.

AR Markers:

The most obvious feature provided by the AR Markers is a means for positioning. By developing a system to recognise the Markers using the Vitas camera, the Markers could then be transformed to be given a position in the games World Space. With each marker transform being used to calculate the positions of all the 3D aspects of the game, the transforms for the markers needed to be stored in a way that they could be efficiently accessed whenever needed. By making the decision to store each of the markers in an array, all of the transforms would be stored in a contiguous block of memory and iterating through or accessing them individually will result in cache hits, instead of potential stalls and cache misses were they stored differently. As the transforms are used so regularly in the state update functions, this could have had a major effect on overall performance. With these world space positions, in game objects could then be rendered wherever there is a Marker. With a system designed for recognising the Markers in place, some of the main benefits of AR could be take advantage of, such as accessibility. As the player now has physical control over the placement of objects in the game world, they are no longer restricted by any awkward joystick control or confusing button pushes and are instead able to move and alter the game however they feel comfortable. This can potentially open the game up to a wider audience offering something not previously available.

Camera:

Adding a gameplay feature that made use of the camera was a priority for this project as it would add an extra dimension of interactivity for the players but also would be a way of fully utilising the available hardware and potential of the PlayStation Vita. A simple but effective way of achieving this was discovered during research. In Knightfall AR (A&E Television Networks Mobile, 2017), a medieval AR Tower defence game which had a mechanic where the player had the ability to target a specific enemy in the game and then have a tower shoot at that enemy, giving the player the feeling that they are commanding a tower to fire at an enemy. By placing a crosshair at the centre point of the camera feed, the player could identify the enemy they wanted to attack, then press the “O” button. A ray would then be cast from the centre of the screen and if the ray collided with an enemy, the target would be set, and the tower would attack it.

Touch Screen:

Whilst the Markers allow the player to control aspects of the game physically, to fully utilise their AR capabilities, the player needs to also interact with the attached rendered game objects. The Vitas touch screen capabilities provide an ideal means for. It is this uninterrupted link between the game world and the physical world that gives AR games an extra layer of immersion and interactivity that isn’t found in other types of games. In this Tower defence game, between rounds, the player can place a new Marker wherever they want and then tap that marker on screen to spawn a new tower or can tap a tower that is already built and upgrade it. The tower that is spawned or which upgrade is added is then chosen by tapping one of the onscreen UI buttons.

Image with Tower selected for upgrade in Red:

A screenshot of a computer game

Description automatically generated with low confidence

GUI:

To provide a way of showing the necessary game data, a GUI class was created. In the GUI class a sprite that represents a pop out window was created. This pop out window would open between rounds and would display to the player; a button to start the next round, if they had a build/upgrade available, and the buttons for each build/upgrade, that would also change colour based on their availability. To create the buttons a Struct called Button was created that made use of composition techniques. This button struct was composed of two Gef::Sprites(one for when pressed and one for when available), a bool and a Gef::Vector4 for position. The GUI class included a health bar in the top right of the screen that showed the players current health. If an enemy made it to the end of the map the players health would decrease, and the health bar would decrease also. The current round number is also displayed in the top right using this GUI class.

GUI with unavailable buttons:

A screenshot of a game

Description automatically generated with low confidence

GUI with available buttons:

A screenshot of a computer game

Description automatically generated with low confidence

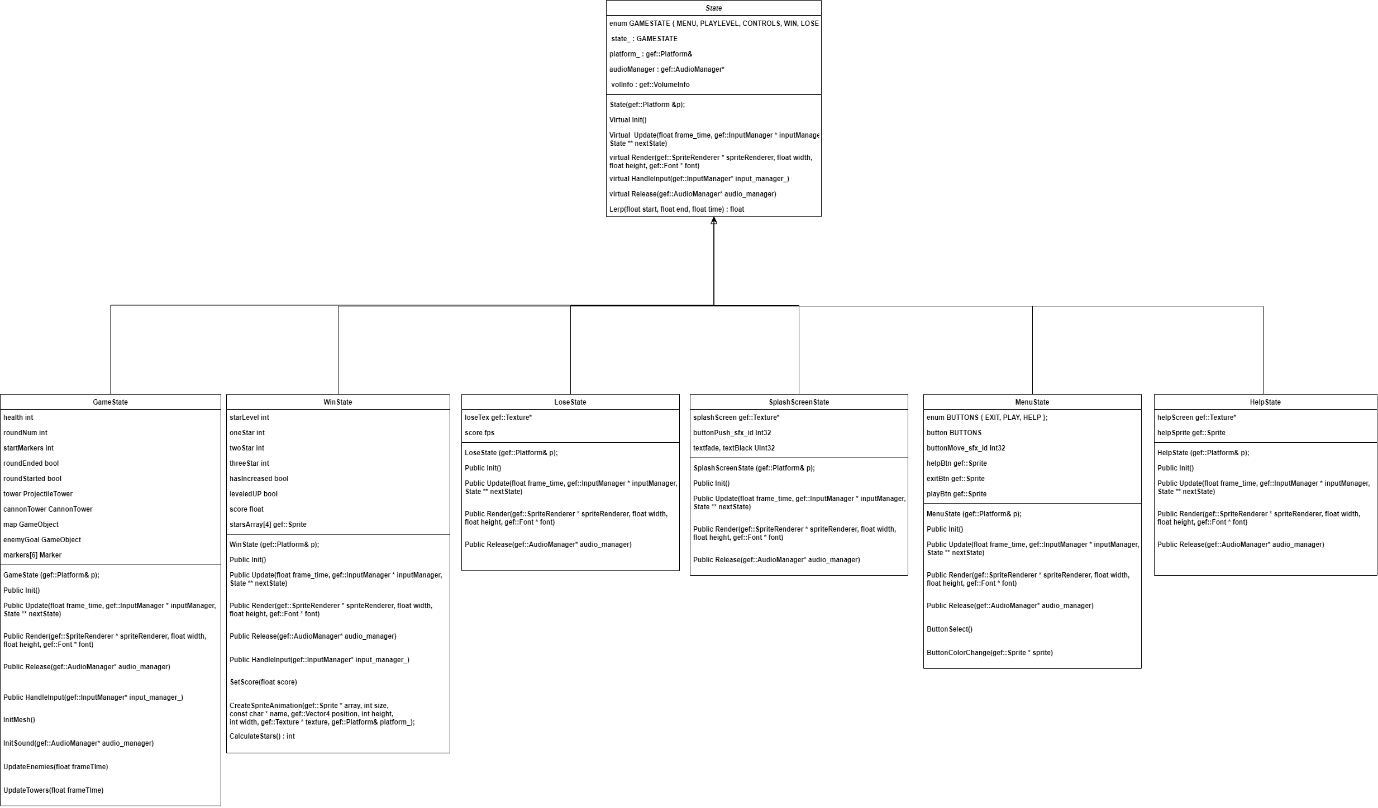
State machine

This game made use of a state machine to change between the relevant states. The game would start in the splash screen state before moving to the Menu state on a button push. From the Menu state the player could choose to exit the game, launch the Game state or go to the Help state. From the Game state the player would be transferred to the Lose or Win state based on their in-game performance. If the “x” button was pushed in the Help, Lose or Win states, the game would return to the Menu state. In the Ar\_App class, the Init, Render, Update and Release functions of the current state would then be called.

Diagram

Description automatically generated

To create the State Machine, a Super class called State was created. In the State class, several virtual functions were created that would be overridden be each of the Child class instances. These virtual functions were the Init, Render, Update and Release functions. Classes for the Menu, Game, Win, Lose and Help states were then created that inherited from the State Super class. To implement the change from one state to another the game made use of static functions. In the Ar\_App class, a State pointer was created called “currentState” that would be set in the AR\_APP Init function. A static function that had a State pointer as a parameter called “SetState” was also created in Ar\_App. With the SetState function being static it could be called anywhere in the game that had access to Ar\_App to set the next State. In the SetState function, the “currentState” release function would be called, then the “currentState” pointer would be set to the State that was passed into the function call as a parameter and the new “currentState” Init function could be called.



Game Object

As the game would need several different objects that would all need a local transformation matrix and a mesh, a GameObject class was created. This class would inherit from the Gef::MeshInstance class so that every instance of GameObject could be easily rendered using the Gef:: 3d Renderer. Using an Update function, the GameObjects’ local transform could be constantly updated throughout the game. With these features of the GameObject set up, child classes could then be created that inherit for the GameObject class, the in-game Towers, Projectiles and Enemies all inherit from GameObject. The game would make use of three different types of towers. The Gate Tower would serve as the entrance to the map where the enemies spawn from and is the tower that the attacks when the player targets an enemy. The Projectile Tower in the game has it position fixed once the game starts and shoots at any enemy that gets within its range. This range is checked by getting a vector from each enemy to the tower itself and comparing the length of this vector with the towers range value. The third tower in the game is the Cannon Tower. The Cannon towers are rendered with a cannon mesh and fire, with a fixed fire rate, cannon balls along the forward vector of the cannon. The player can rotate and move the cannon around the map to fire in any direction they desire. Due to performance limitations, the cannons only check for collisions between the cannon balls and the nearest enemy at the time of firing.

Enemy:

As the Enemies in the game needed to move through the map, a way for each enemy GameObject to know where to go needed to be implemented. To do this, a function was created that is called in the Game states Init function that reads values from a .txt file. This function would read each element of the file and return it as a char pointer. A separate function was then created to convert the char pointers to a single string and looped through each element of the string. The elements of the string were then converted into floats using their ASCII values and stored in a vector of floats. Finally, this vector would be iterated through converting every three floats into a Gef::Vector4 and storing it in a vector of waypoints that is passed to each enemy in its Init function. An int value is used to assign the enemy’s current waypoint. The enemy class has a Gef::Vector4 called velocity that is set by calculating the vector between the enemy’s current waypoint and the next waypoint and normalising the result. This velocity vector is then multiplied by the enemy’s speed variable and added to the enemy’s position to create the enemy movement behaviour.

Screenshot of Level showing Enemy Start and End points:

A model of a castle

Description automatically generated with low confidence

Graphical user interface

Description automatically generated with medium confidence

**Reflection & Evaluation:**

As the development of this project would be largely disrupted by the current global situation and access to the required Hardware would be limited, a strict schedule for the project had to be created and adhered to in order to meet the standards expected and the project deadlines. This involved having to work on any non-AR/Hardware specific functionality of the game at home in order to maximise the time spent on campus with access to the Vita. This created a degree of pressure not felt in previously undertaken modules but given the scenario and the game implemented, following this development strategy was the correct course of action.

Upon first completion of the game, it felt too simple and to distinguish it from a tech demonstration, elements to improve game feel were added. Background music for each state to add more life, sound effects were added for when the cannons or archery towers fire, as well as when the round starts, and enemies die. Screen Shake was added whenever an enemy reached the end of the map. A Sprite animation was added to the Win state to show the player how well they did. Based on the amount of health the player has left at the end of the game, a star rating out of three was assigned and the animation would cycle through three stars to convey the players performance to them. Whilst sprite animations were added, a further step that could have really enhanced the feel of the game would have been to add a walking and dying animation to the enemy GameObjects but as it was deemed not to be within the scope of the module, was left out of the game.

Whilst the concept of the AR Markers is a functional way of implementing AR and is an impressive piece of technology, it has a few limiting factors that can lead to an unplayable experience. The 0.3-megapixel camera coupled with the weight of the Vita being held in one hand, as the other needs to modify a marker for in-game purposes, often leads to a shake or jitter by the player that causes the image processing to lose track of the Markers. This was a regular occurrence during development and testing. When this occurs, any GameObject rendered on a lost Marker would freeze in its Screen space position until the Marker was found again. Two solutions were devised to attempt to solve this issue. The first was to simply stop rendering the GameObjects attached to a Marker that has been lost. Whilst this would solve the issue of the GameObjects sticking to the screen, it was not clear that an error had occurred tracking a Marker and seemed more like the GameObject had just disappeared. Instead a second solution was implemented where if a Marker that was being used was lost by the camera, the game would pause, and an error message would pop up on screen until it was found again. Neither solution is ideal however, as they can’t solve for losing track of the Markers. The first solution could cause the player to lose the game by not rendering in an GameObject and the second solution can disrupt the players immersion. Upon completion of this project, it became clear that the reliance on the Markers would be the limitation to any success of this technology.

An alternative to the Markers would be to use HMDs. Microsoft has long been trying their hand at AR with their own HMD, HoloLens and now HoloLens 2. At E3 in 2015, Minecraft was presented as a playable game on the HoloLens, with the player being able to render and interact with the Gameworld on a table on stage. Whilst the HoloLens presents an exciting avenue for AR Game development, the price and availability will be stumbling block for consumers and hinder any success until addressed. This has led to the opinion that AR has the potential to be a massive market for games but not in either of its current implementations.

The aim from the outset of this project was to research an emerging technology and develop a game that can take advantage of that chosen technology. By developing a game that makes use of real time movement of the AR Markers with the Cannons, as well as providing a way to set the position for other towers, and by utilising the available hardware of the vita with the touch screen to provide an uninterrupted link with the game world and camera to target specific enemies in game, this tower defence game demonstrates that development of this type of game using this technology could be worthwhile in the future.

# References

A&E Television Networks Mobile, 2017. *Knightfall™ AR.* s.l.:A&E Television Networks Mobile.

Argaman Creative, 2018. *Master of Towers - AR Tower Defense.* s.l.:Argaman Creative.

Cummings, B., 2019. *TRACER: Sailors Use Augmented-Reality to Train for Combat,* https://www.onr.navy.mil/en/Media-Center/Press-Releases/2019/ONR-Global-TechSolutions-TRACER: Office of Naval Research Science & Tecnology .

Evangelista, A. et al., 2020. Unveiling the technological trends of augmented reality: A patent analysis. *Computers In Industry,* 118(June 2020).

Huang, S.-H., yang, Y.-I. & Chih-Hsing, C., 2012. Human-centric design personalization of 3D glasses frame in markerless augmented reality. *Advance Engineering Infomatics,* 26(1), pp. 35-45.

Lego, 2020. *Hidden Side.* [Online]   
Available at: https://www.lego.com/en-us/themes/hidden-side/about  
[Accessed 10 1 2021].

Next Games Corporation, 2012. *The Walking Dead: Our World.* Helsinki: Next Games .

Niantic, I., 2016. *Pokémon GO,* California: Niantic, Inc..

Rothmann, D., 2019. *Enhancing AR with machine learning.* [Online]   
Available at: https://towardsdatascience.com/enhancing-ar-with-machine-learning-9214d2da75a6  
[Accessed 6 1 2021].

Statistica, 2019. *Number of smartphone users worldwide from 2016 to 2021.* [Online]   
Available at: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/  
[Accessed 5 10 21].

The Middle Gray, 2017. *Castle Must Be Mine.* s.l.:The Middle Gray.

Wetzel, R., McCall, R., Braun, A.-K. & Broll, W., 2008. *Guidelines for designing augmented reality games.* Toronto, ACM - Conference on Future Play : Research, Play, Share.