CMP404: Applied Games Technologies

Dylan Black | 1403307

Word Count: 3256

Table of Contents

1. Abbreviations

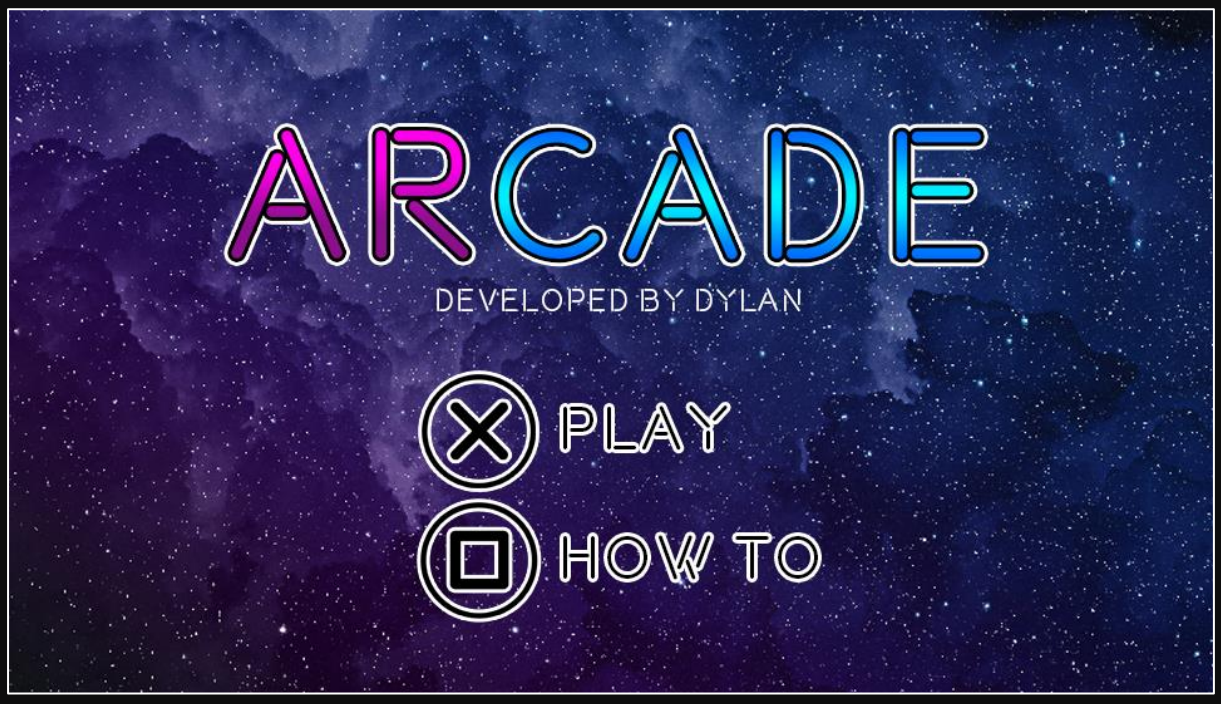
Full Words	Abbreviations
Markers 1,2,3,4,5,6	M1, M2, M3, M4, M5, M6
Augmented Reality	AR
PlayStation Vita	PSV
Frames Per Second	FPS

2. Guide

Direct the player from start to finish using gravitational orbits while avoiding the patrolling enemies!

To control the player, AR markers M3-M6 must be placed to force the player to orbit around them, directing their path. Before the player enters a marker radius, buttons L2 or R2 must be pressed to define either left or right orbit, respectively. Once the player enters an orbit radius, pressing the cross button will lock them into orbit. Once locked on, the player will continue their path around the orbit marker until the cross button is pressed again – releasing them in the desired direction (towards another orbit radius). The player only has three lives, so proceed with caution!

Load the application and press either cross for "PLAY" or square for "HOW TO". It is recommended to run through the "HOW TO" screens first before playing:



2.1. How To

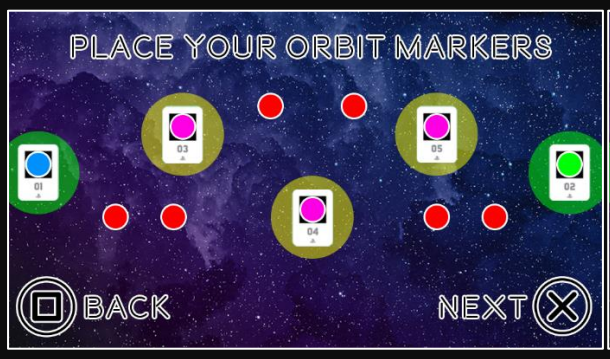
Selecting "HOW TO" will advance to the first of several screens describing gameplay.

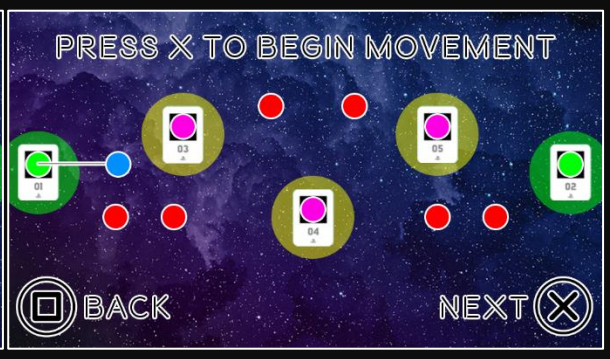
The first displays a basic description of the game, the second defining a key for the proceeding screens:

After viewing the first two "HOW TO" screens, the user is presented a detailed walkthrough of gameplay, beginning with the placement of the start and finish markers (M1 & M2).

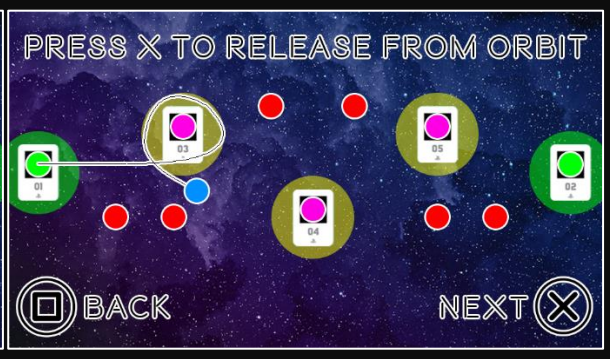
Once these markers have been placed, the application will then build the level around both points by placing the player on the start location and pseudo-randomly spawning enemies between the two:

Next, the user is shown where the orbit markers could be placed in the provided example and then, informed how to begin moving the player:

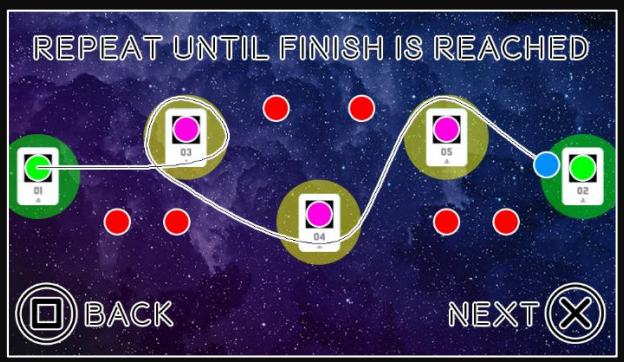




The following screens explain how to control the orbit of the player both with a text and visual description:

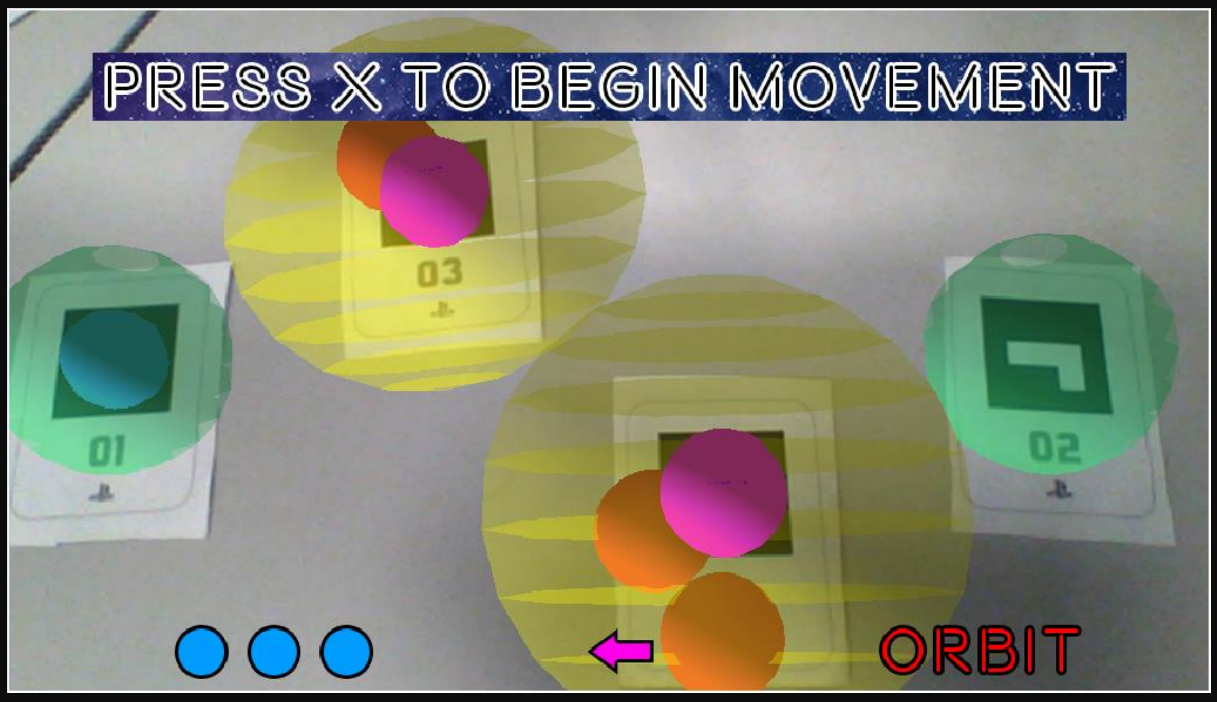


The final screen “HOW TO” shows the player’s remaining course throughout level. These screens have been designed to visualise the entire process of gameplay:



2.2. Play

It is expected that the user has completed the “HOW TO” series before selecting “PLAY”. The user is presented with the camera displaying whatever is in front of the PSV console. The user should place their start/finish markers and continue the process described via the “HOW TO” series. The level can end one of two ways – losing by colliding with enemies more than three times or winning by reaching the finish marker.



The level includes a simple HUD to inform the player of three features – remaining lives, orbit direction and if the player is set to lock onto an orbit:

The user can press the start button at any point to restart the level. This feature has been implemented in case the player unintentionally travels too far out the view space. The user can also press triangle to reset the lighting in the scene for any reason which also resets the marker tracking.

3. Application Features & Operations

Various gameplay features have been implemented within the application and this section will detail each one in order of use. A diagram outlining the application process can be found at APPENDIX A.

3.1. Start & Finish (M1-M2)

The start and finish points for the level are rendered as semi-transparent green spheres and are defined by the positions of M1 and M2. These two markers can be placed as far apart or as close together as possible. Once these markers have been placed and the user is content with their locations, the cross button will allow the application to proceed to the next stage: building the level.

3.2. Level Building

The position of M1 is used to assign the starting position of the player, all player movement afterwards is localised to this position. The distance between M1 and M2 is then calculated and used to define (a) the starting positions of all enemies and (b) the maximum number of enemies to spawn.

3.3. Player

The player is rendered as a blue sphere and makes use of simple sphere collision detection to detect when they hit enemies and reach the finish point. For movement, the player uses its right vector as its forward vector and constantly moves in its direction at a speed of 2.5 multiplied by the current FPS.

3.4. Enemies

Enemies are rendered via red spheres and contain an un-rendered radius sphere to control their movement. They are updated via their designated direction (1,2,3,4) representing up, down, left and right respectively. The enemy is applied a velocity of 0.001 in whichever direction they are currently heading and should always be colliding with their radius. If the enemy moves out with their radius (they fail the collision check) their direction is reversed, and they continue moving at 0.001 in the new direction. This process is repeated to simulate back-and-forth movement.

3.5. Orbits (M3-M6)

Orbits are rendered using pink spheres and have a larger, semi-transparent yellow sphere surrounding them representing their radius. For the player to achieve a successful orbit, they must first pass the collision check with the orbit radius. Depending on the defined turn direction of the player (left or right), the player's right vector is gradually rotated using an amended function from the Unreal Engine documentation (Unrealcpp.com, 2017) at an angle value of either -0.75 or 0.75, respectively.

3.6. Error Management

At any point during either the level building process or actual gameplay, if M1 and M2 aren't visible, the entire application will pause and prompt the user to return the markers to the view space.

4. Technological Design

This section will discuss the theory behind the design of the application technology.

4.1. Developing Applications for Augmented Reality

A publication included in the book, “Handbook of Augmented Reality” (Furht, 2011), details various models of designing games for AR (Wetzel et al., 2011). The chapter continues from the book by introducing the main elements involved in a game focussed on AR technologies. These features include integrating physical locations and objects into the gameplay, forcing the player to interact with the world around them. The text explains these attributes compel the player to use established emotions, thoughts and real-world understandings to fully exploit the concept of using pre-existing resources to deliver a completely immersive game world. Even though AR technology is among the greatest advances of computer science to date, it continues to confuse its first-time users. This is one of the biggest problems a developer must face when developing an AR application.

4.2. Guidelines for Augmented Reality Game Design

To develop a successful PSV AR application, the project used the journal’s “game design guidelines” as a basis when determining if the features and operations were suitable for AR implementation. The first listed rule in the chapter states, “justify the use of AR”. This is important as an application that can be run without AR technologies might as well be developed for non-AR software. The application validates using AR as the game cannot be played without moving real-world AR markers. The second rule expands on the first, “engage players physically”. For an AR game to be successfully developed in the context of the technology, the user must physically interact with either real-world environments or the rendered objects. The PSV AR application requires the help of the user when determining the structure of the game level by placing start/finish markers and the placement of orbit markers to allow gameplay. An example of an immersive experience utilizing the functionalities of AR within a game environment is, arguably one of the most popular mobile AR games ever developed, Pokémon Go (Niantic, 2016). The game requires its users to physically travel areas of the real world in search of Pokémon to collect. In order to fully immerse the player within the levels, the PSV AR application has been designed around the image recognition and physical movement of AR markers. Resuming the list (Wetzel et al., 2011), the section outlining specific technology and usability guidelines is most relevant to this project. The list expresses, “keep the interaction simple”. This has been a defining element when designing the PSV AR application as an extensive array of control schematics or physical interactions would only overwhelm the user even more. With only a few main buttons being pressed on the console during gameplay (cross, L2, R2), the user is expected to rely on their hand-eye coordination to help the application construct the level and to simulate gameplay. Another main point the chapter makes is to “take display properties into account”, which the PSV AR application does by minimizing the size of the available AR markers to involve as many of them as possible. The restrictions made by the PSV can easily be interpreted as flaws, rather than the application purposely making use of what the console provides in terms of view space and image recognition. Therefore, action has been taken to use as many functionalities of the PSV AR software as possible given the limited space the camera can display.

5. Innovation

This section will detail potential areas for innovation in the application, using AR technology.

5.1. Expanded Play Area

For the PSV AR application to further exploit AR technology, the play area could be expanded over an entire room – rather than a table top. This was the initial concept of the application, although due to physical restraints on the console the implementation was made impossible. This innovation method would require the user to place AR markers at different locations around the room and to physically move with the player to keep them in the view space. Enemies would spawn in a similar fashion to their final implementation and the orbit markers would too. The idea stemmed from research around the popular mobile AR app, Pokémon Go (Niantic, 2016). Further research discovered the previously mentioned study (Wetzel et al., 2011) that frequently refers to the AR game, TimeWarp 2. Within this game, users had to explore the real-world to traverse through different time periods in search of stranded gnomes. The entire concept of contextual AR games is extremely appealing to a developer as it involves the user in both the game and real world like no other current technology.

To efficiently expand the play area, development would be focussed on scene mapping majority of the room, at different points of the game. Firstly, the game would invoke the user to place the starting marker via both scene mapping and marker recognition. Then, the finish point would be randomly generated at a given distance, continuing with pseudo-randomly spawning enemies between the two points. The user would then be expected to place their orbit markers around the room to simulate their desired path. Once happy with the locations of all markers, the application would begin moving the player by pressing the cross button – like the current application. The program would have stored the location of the first marker and the rest of the game world would be based off this position. The user would follow the player by walking with the PSV to keep them in the view space, pressing the cross button again to lock-on/release from orbit to reach the finish point. The current application makes full use of the provided AR markers, although an early decision in development discarded the implementation of scene mapping due to significant drops in frame time and the restricted view space.

5.2. Multiplayer

Another innovation method which could deeply enhance the playability of the application would be the introduction of a multiplayer mode. This could involve the game being run on several different PSVs connected over a network or the use of real-life equipment. For example, handheld paddles or a physical tracking system which would allow more than one user to stand around the room, replacing the orbit markers. As the study (Wetzel et al., 2011) describes TimeWarp 2 in increasingly greater detail, it is explained that the game utilizes hardware known as an “Ultra-Mobile PC” (UMPC). Two users both have their own UMPC and must work together as they perform different roles. This concept could be applied to the PSV AR application by requiring its users to work in a team to construct an orbital path for the player to reach the finish.

6. Software Design

The application has been designed with an object-oriented programming style in mind.

6.1. Source

The primary controller of the application is made up of the “ar_app” class. Within this, each application state is initialised, updated and rendered individually depending on the currently active state.

6.2. Application States

Every application state inherits from the main, “base_state”. This class includes only a header file and defines virtual functions for its children to override. Local pointer objects are also specified here including a texture vector, platform, sprite renderer, primitive builder, Sony controller and a current state variable.

6.2.1. Informative States

The “splash_state”, “howto_state” and “endgame_state” are all primarily focused on displaying descriptive information to the user. The Splash state acts as the main menu of the application, the How To state explains the gameplay mechanics and the End Game state deciphers whether the user has won or lost the game, depending on the state passed in from the Game state.

6.2.2. Game State

The bulk of the application is encapsulated within the “game_state” class. Game objects such as the player, enemies, orbits, start and finish are declared within its header and defined on initialisation. The camera image feed is also setup within this class and rendered appropriately. During its update process, the tracking library is used to detect AR markers within the view space. The marker limit (6) is looped through and if the current marker is found in the scene, its transform is stored in a separate vector. Each marker is allocated an object(s) and these objects are only updated/rendered if their corresponding marker is visible. The entire process of updating the individual objects is paused if M1 and M2 exit the play area. At the end of its update, the “collision” class is used to detect collisions between objects. If the collision check returns true between the player and any enemy, the player lives are decremented until equalling zero – this causes the state to switch to the “endgame_state”, defining the user has lost the game. Otherwise, if the player collides with the finish marker, the state is switched to the winning “endgame_state”. If the player collides with any of the available orbit radii while set to allow orbit, they will begin rotating around the object via the defined turn value.

6.3. Game Objects

All game objects within the application inherit from, “game_object”. This class defines global variables and functions used across all game objects – such as manipulating its transform, applying velocity, scale, rotation and translation values and setting the material of the object.

6.3.1. Player Object

The “player_object” consists of logic to determine whether they can lock-on/release from an orbit, a definitive turn amount variable and a controller to decide if they can die or are immune from damage (for 3 seconds). The sphere object representing the player is defined in the class constructor.

6.3.2. Orbit / Enemy Objects

The “orbit_object” and “enemy_object” both include a separate “game_object” which acts as their radii. The primary and radius sphere objects are also defined in the class constructor.

6.4. Tools

Two separate classes are used within the application acting as tools to apply different elements.

6.4.1. Collision

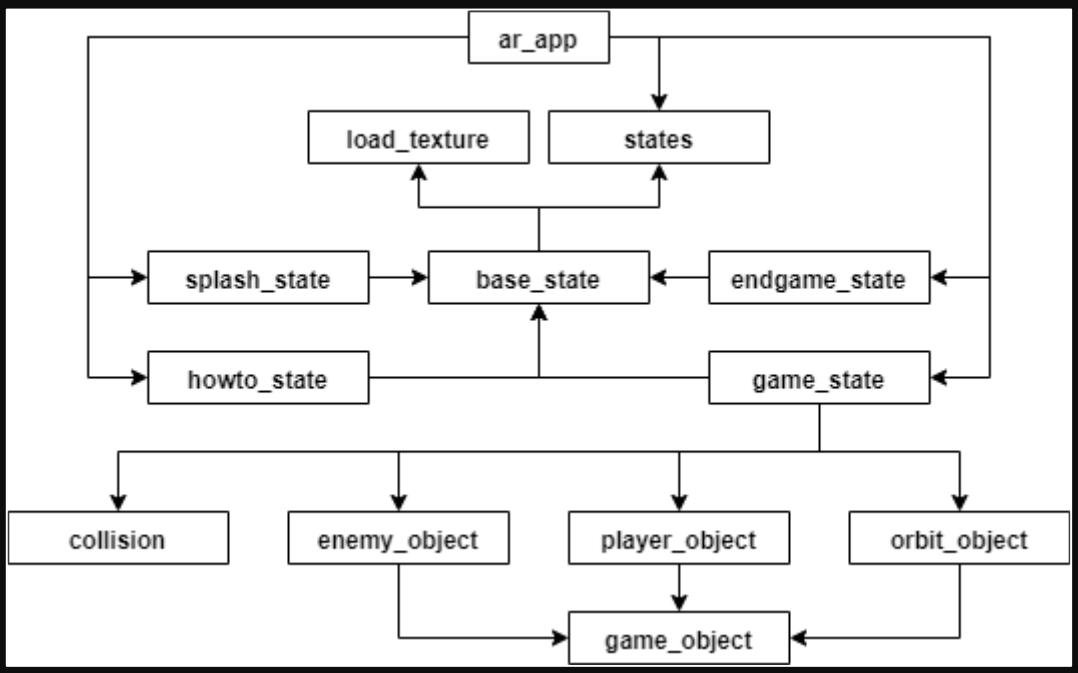
The “collision” class contains functions to detect both sphere and AABB collision between meshes.

6.4.2. Load Texture

The “load_texture” class contains one function which simply creates a texture from an external PNG.

6.5. Class Diagram (Summarised)

Below is a summarised UML diagram of the application:



An in-depth visualisation of the class diagram can be found in the attached file: “ClassDiagram.png”.

7. Reflection

In hindsight, the final application could be improved by including/polishing several features.

7.1. Scene Mapping

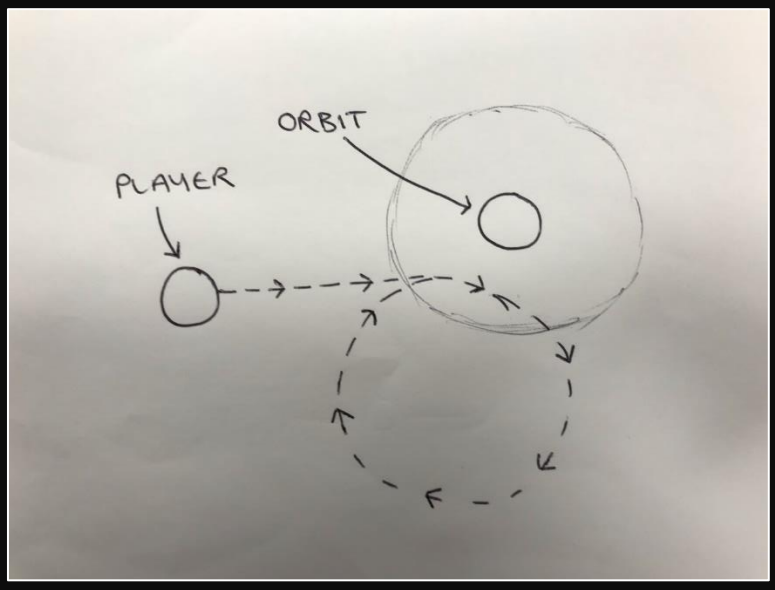
The use of scene mapping would have allowed the application to recognize appropriate start and finish points without using M1 and M2, this would leave these markers to be used as orbits instead. This feature would be appropriate if an acceptable workaround for the frame rate drop is implemented.

7.2. Orbit Locations

The application would have benefitted from a feature to store the orbit location on a button press, and then allow the user to reposition the same orbit marker to store a separate orbit, and so on. This feature would minimize the need for using most of the markers and potentially allow the remaining markers to be used as other game objects such as walls able to bounce the player in another direction.

7.2. Orbit Mechanic

Currently, the user can press buttons L2 and R2 to apply either a negative or positive turn value, respectively. This mechanic can easily be exploited in terms of entering an orbit radius and directing the player away from the orbit marker – a visualisation of this process can be found below:



Many different theories were implemented to simulate the player properly orbiting around the marker. For example, by multiplying the player's local transform with the orbit transform instead of M1. However, this revealed various other problems and it was decided to implement a more basic mechanic instead. Further development would see this feature implemented as initially planned for a more suited, gravitational mechanic.

7.3. Orbit Movement

Another feature which failed to reach the finished application was the ability to move the orbit markers after the player has locked into its orbit. This feature would have both pros and cons, with the main benefit being to fully immerse the user within the game world by having them constantly moving the marker positions to direct the player. Although, by allowing the user to do this, they could potentially cheat the game and simply move the current orbit marker towards the finish point, skipping the challenge of avoiding the enemies and heading straight for the end game.

8. Problems & Solutions

Many obstacles were encountered during development of this application and this section will detail the ones which had the most impact on the final version.

8.1. Orbit

The game has been named, “ARCADE” for a couple reasons. The first being the obvious wordplay between AR and arcade. The second reason is primarily focussed on the gameplay, as the initial plan for the application was to simulate retro 8-bit mechanics as seen in games such as Pac-Man (Namco, 1980) in which players explore a maze using 4-directional movement. The idea for the orbit mechanic within the PSV AR application included four paddles situated at the four corners of the marker used to bounce the player in the direction the paddle is facing – simulating 4-directional movement. This mechanic was successfully implemented, although it proved difficult for the player to directly hit the paddles on every try. Several attempts were made to help direct the player towards the paddles by adding more, un-rendered paddles for the player to bounce off. However, this resulted in the entire mechanic becoming a little too complex and was eventually scrapped for the much simpler orbit simulation included in the final application.

8.2. View Space

Using the default resolution of the AR markers forced the view space to become over-crowded and made the game extremely difficult to beat with the size of the enemies. This could have been combated by simply decreasing the size of the rendered objects, although the physical size of the AR markers frequently caused issues when positioning the orbits as they would overlap with each other. The final solution devised for this problem was to print off a smaller set of markers to easily move them around the view space without restrictions.

8.3. Scene Mapping vs Marker Recognition

As mentioned in previous sections, the initial plan for this application was to use both scene mapping and marker recognition. In early development, two separate projects were created to evaluate the benefits of using scene mapping with marker recognition against just marker recognition. The scene mapping project was favoured in the beginning as the intentions were to utilize a much larger view space, although the frame rate was cut in half compared to the marker recognition project. As development continued, the scene mapping feature became obsolete as the view space was restricted and didn’t require the use of scene mapping to register a large space. Therefore, the scene mapping concept was added to the list for further development and work continued the project using only marker recognition.

9. References

9.1. Literature

1. Furht, B. (2011). *Handbook of Augmented Reality*. New York: Springer. [online]. Available at: https://link.springer.com/chapter/10.1007/978-1-4614-0064-6_25.
2. Liu, Y., Dong, H., Zhang, L. and Saddik, A. (2018). Technical Evaluation of HoloLens for Multimedia: A First Look. *IEEE Multimedia*. [online]. 25(4), pp.8-18. Available at: https://ieeexplore.ieee.org/abstract/document/8493267.
3. Pokémon Go. (2016). *Niantic*. Mobile app. Available on iOS or Android via Apple App store or Google Play store.
4. Pac-Man. (1980). *Namco*. 8-bit arcade game. Available to play online at: http://www.freepacman.org/welcome.php.
5. Unrealcpp.com. (2017). *Unreal C++ | Rotate Angle Axis*. [online]. Available at: https://unrealcpp.com/rotate-angle-axis/. [Accessed 3 Dec. 2019].
6. Wetzel, R., Blum, L., Broll, W. and Oppermann, L. (2011). Designing Mobile Augmented Reality Games. *Handbook of Augmented Reality*. [online]. pp.513-539. Available at: https://link.springer.com/chapter/10.1007/978-1-4614-0064-6_25.

9.2. Assets

1. Bastien Sozoo. *Beon*. [font]. Available at: https://www.1001fonts.com/beon-font.html.
2. Wallpapermaiden.com. *Space Background*. [image]. Available at: https://www.wallpapermaiden.com/wallpaper/30797/galaxy-shiny-stars-universe-outer-space/download/3000x2000.

10. APPENDIX A

Application process diagram: