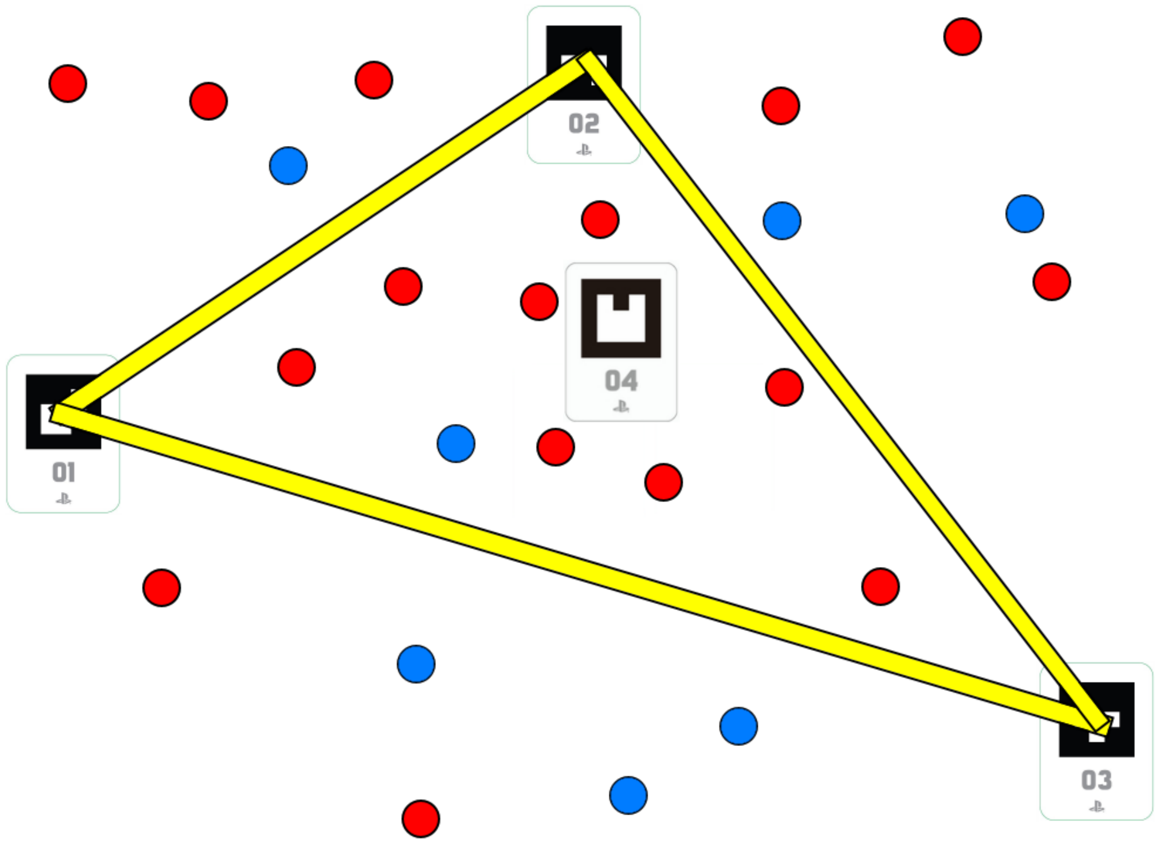
Advanced Gameplay Mechanics
Greg Balbirnie

Game Overview

The application that has been created uses three of the augmented reality (AR) markers which create electrical towers that create a triangle with electric beams co nnecting the three. Several enemy characters will move around the table and when a button is pressed, every enemy within the triangle will be killed, however some characters will be friendly and so the user must avoid killing those creatures. The player will receive points for each enemy killed and lose points for every friendly character killed and must attempt to accumulate the most points by the end of the time limit.



Research/Inspiration

The Nintendo 3DS (Nintendo, 2011) comes with several augmented reality cards featuring characters from various Nintendo franchises. These cards can be used in built in AR games such as a diorama mode in which a 3D model of the character represented on the card is rendered above the card and a fishing game in which the user can use their 3DS to catch fish in a pond rendered on the AR marker. An interesting feature of this fishing game is that the type of fish in the pond is determined by the colour of the object that the marker is placed on. This is a feature that could not be replicated without the use of AR technology as the use of the camera is what makes the game able to gather information about the surface the player is playing the game on.

“Pokémon GO” (Niantic, 2016) is a popular mobile game with an augmented reality feature. With AR turned on, the Pokémon that the players can catch in the game are rendered over their phone camera feed as if they are present in the real world. The models will also move if the camera is moved to place them in a realistic position.

The game “EyePet” for the Playstation 3 (SCE London Studio, Playlogic Game Factory, 2009) uses a camera as well as a marker to render multiple objects atop a feed of the user, with the main objects being a digital pet. This pet is able to move independent of the marker by finding the transform of the marker on the player’s floor when the game is started and then other objects, such as the pet’s food. This game features an excellent use of marker-based AI and by using a stationary camera, is not reliant on the marker for all objects in the game.

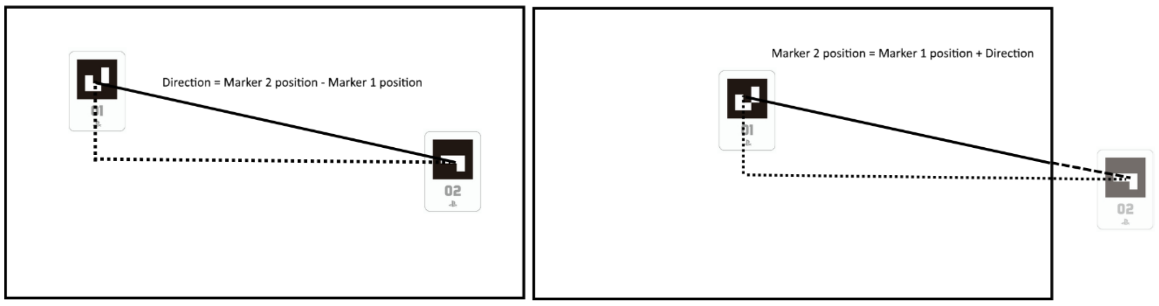
Design Justification

Marker Usage

Three markers are used for the electrical towers and beams as this is the minimum required to make a two-dimensional area and there was concern that if the user was able to use more than three markers the game would become too difficult to play and the attack would cover too great an area. A fourth marker I used to denote the centre of the playfield from which the characters are positioned and it is not to be moved for the duration of the game.

Position Interpolation

In order to ensure that all three markers are rendered, the application uses the transforms of all visible markers to interpolate the position of the off-screen markers. Using the markers allows the electrical pillars to be moved around freely with very little difficulty, making it easily accessible for players regardless of whether or not they are avid game players. Another benefit of the AR markers if that the game can be displayed as if it were on the player’s table, which is fun, especially for young players and could not be done with AR.


Direction = Marker 2 position - Marker 1 position
01
02
Marker 2 position = Marker 1 position + Direction
01
02

Characters

There are two types of characters in the game, enemy creatures and friendly creatures. The creatures move around and act in the same way however the two types can be easily distinguished by their colour with enemy characters being red and friendly characters being blue. These colours were chosen as they are easily distinguishable even for people with colour-blindness. More enemy characters are spawned than friendly characters as the aim is to avoid killing friendly creatures and so the game may be frustrating if the friendly characters spawned too often. The characters will render on the table but not directly on any of the markers and so in order to place the characters in the correct location a similar method to the marker position interpolation is used. The direction from a centre marker and the location each character should be is stored and the characters are offset from the centre marker to the desired location.

Model Loading

In order to make the game more interesting, model loading has been used so that the electrical towers and characters have 3D models instead of just being represented by coloured cubes. As the electrical beams are constantly being stretched and moved, there is no specific object that would make the most sense and so it is still represented by a transformed cube. As the aim of this project is to demonstrate the potential of AR in a video game setting, original 3D models were not created and instead, free 3D models were sourced from TurboSquid (TurboSquid, 2000).

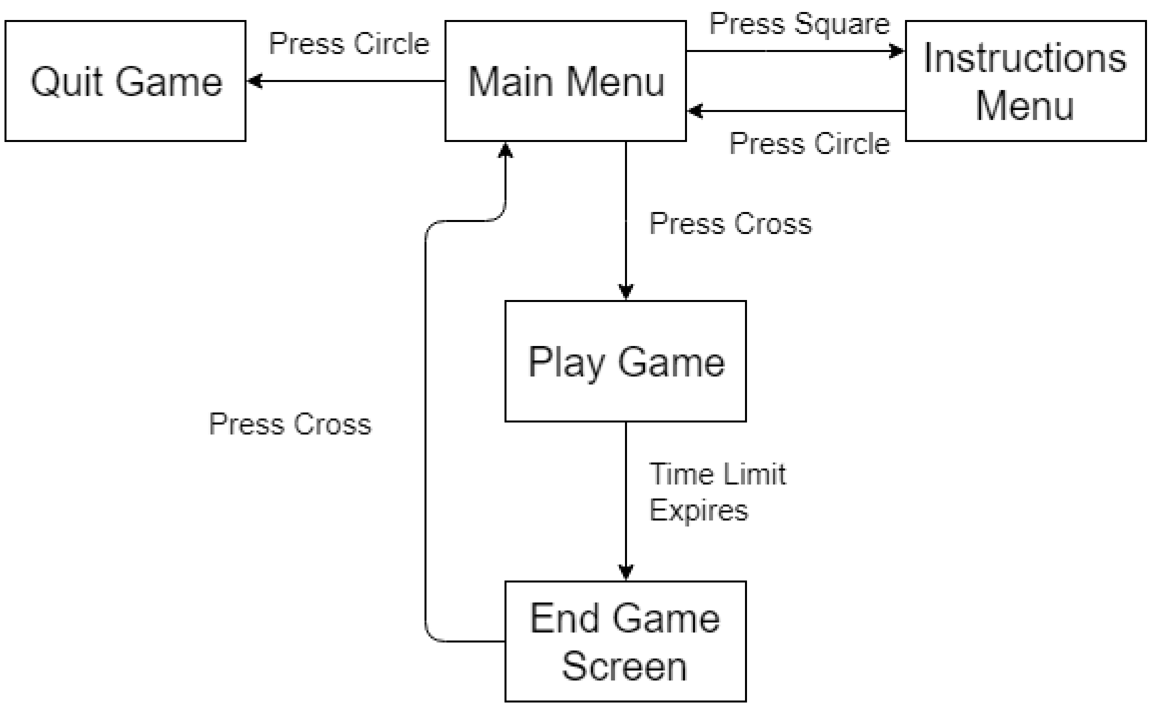
Image Loading

Image loading was used in order to make the game look more appealing as well as convey information to the player. Both of the 3D models that were sourced included textures which are loaded into the project and are applied to the models. The bomb object came with a black bomb texture which was edited in Paint.net (dotPDN, LLC, 2004) to be blue and red, which are more appropriate for the game. Images were created to be displayed as full-screen sprites as the loading screen, start menu, instruction screen and end screen which explain to the user how to play and what to do in order to navigate the different parts of the game.

Audio

Another aspect of the game that was added to improve user experience was the addition of audio in the form of a theme song and a sound effect for the electrical beam. The music was sourced from Bensound (Bensound.com) and the sound effect was sourced from FreeSound (Music Technology Group, Universitat Pompeu Fabra, 2005)The music begins playing as soon as the game is started and the sound effect triggers whenever the user presses cross whilst in the game state.

State Diagram



Game Instructions

When the application is run the start screen will appear and the user will be prompted to either press the cross button to start the game or press the circle button to closer the application. Upon starting the game, a one-minute timer will begin and the user has to arrange the markers so that the triangle of electrical beam is containing the most enemy creatures possible. Whenever the cross button is pressed during the game, all creatures within the triangle are killed and the user will gain points. The aim is to collect the most points within the time limit and when the time limit is over, the end screen is shown with the user's final score. The user can then press the cross button to return to the start screen and either play again or quit.

Software Design

Beam Creation

In order to create the beams of electricity from each tower a cube is spawned halfway between each pair of towers and then stretched between the two. This is done is several stages:

- The difference in the horizontal and vertical positions of each pair of towers are found and using them the angle between each pair of towers and the distance between are found.
- The midpoint is found by adding the two positions together and halving the result.
- An offset matrix is created in order to offset the cube from the markers.
- The Y scale of the offset matrix is set to the distance between the markers.
- The rotation is set to the angle between the two markers.
- The translation is set to the midpoint.
- The cube's transform is them multiplied to the offset matrix, resulting in the desired transform.

This updates each frame so that when the markers are moved the beam always spans the two towers

1. Find the three markers.
2. Find the local transform of each marker by multiplying their transform with the inverse of the centre marker's transform.
3. Calculate the midpoint between each pair of markers.
4. Calculate the distance between each pair of markers.
5. Calculate the angle between each pair of markers.
6. Set an offset matrix to move the beam cubes to the matrix, rotate them to the angle between the two markers and scale the cubes to the distance between the markers in one axis.
7. Place the electrical tower object on the marker.

Marker Interpolation

When one of the three markers is out of view of the camera, the beam will remain where it is expected to be. This is because when the camera can see any pair of markers, the program will save the direction vector between them, then when one goes off-screen the application searches for one of the other visible markers and them assumes the invisible marker is still in the same place relative to the visible marker. The application can therefore place the tower and beam relative to the visible markers making the game far easier to use.

1. Check which of the makers can be seen and save in a Boolean.
2. Loop through each pair of markers in in a nested for loop.
    - 2.1. Check if both markers are visible.
        - 2.1.1. Calculate and store the direction vector between each pair of markers in either direction.
3. Loop through each pair of markers in in a nested for loop.
    - 3.1. Check if one marker is visible and the other is not.
        - 3.1.1. Add the direction vector from the visible to invisible marker to the visible marker's position to find the predicted position of the invisible marker.

Game State Classes

In order to keep the code separated into relevant sections and prevent the project from becoming cluttered, classes were created for each of the game states. Each class contains an update and render function which can be called from the main update and render functions using a switch statement. The current class is represented by an enumerated type and can be changed when specific requirements are met. When the game state is changed the initialise function of that state is called so that when the user restarts the game the game's variables are reset.

1. Set the first state
2. Initialise the state
3. Loop until game is quit
    - 3.1. Check the current state
        - 3.1.1. Update the current state
        - 3.1.2. If the state has been changed
            - 3.1.2.1. Initialise the new state
        - 3.1.3. Render the current state

Characters

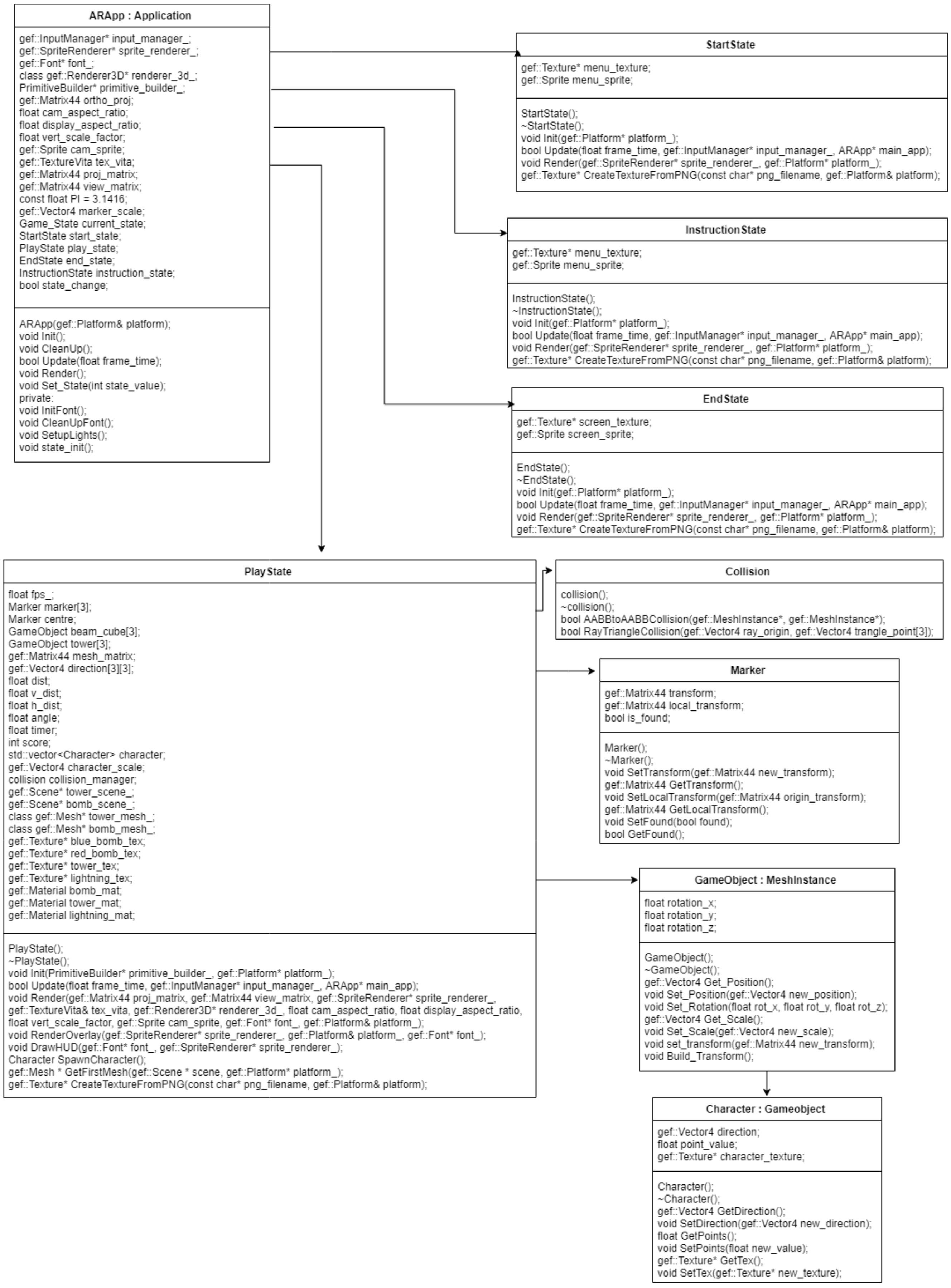
As the two types of characters are largely identical to one another, a character class was created containing details of their position, score value. The enemy and friendly characters are both instances of the character class however they both have a coloured texture and reward a different number of points to the player. The characters are then offset from the centre marker.

```
1.  Check if the centre marker is visible.
    1.1.  Loop for all active characters.
        1.1.1.Store the direction vector from the marker to the character.
2.   Check if the centre marker is visible.
    1.1.  Loop for all active characters.
        1.1.1.Translate the character by the direction vector from the marker to the character.
```

Player Triangle Collision

The algorithm for checking if the characters are within the triangle is based on the Möller–Trumbore intersection algorithm (Möller, Tomas; Trumbore, Ben; 1997). The direction of the ray is always considered to be in the positive Z axis as the collision should always be checked directly where the character is standing. The triangle for the algorithm is defined by the positions of each of the markers and the algorithm is able to calculate vectors defining the beams based on these points.

Class Diagram



```
ARApp : Application

gef::InputManager* input_manager_;
gef::SpriteRenderer* sprite_renderer_;
gef::Font* font_;
class gef::Renderer3D* renderer_3d_;
PrimitiveBuilder* primitive_builder_;
gef::Matrix44 ortho_proj;
float cam_aspect_ratio;
float display_aspect_ratio;
float vert_scale_factor;
gef::Sprite cam_sprite;
gef::TextureVita tex_vita;
gef::Matrix44 proj_matrix;
gef::Matrix44 view_matrix;
const float PI = 3.1416;
gef::Vector4 marker_scale;
Game_State current_state;
StartState start_state;
PlayState play_state;
EndState end_state;
InstructionState instruction_state;
bool state_change;

ARApp(gef::Platform& platform);
void Init();
void CleanUp();
bool Update(float frame_time);
void Render();
void Set_State(int state_value);
private:
void InitFont();
void CleanUpFont();
void SetupLights();
void state_init();
```

```
StartState

gef::Texture* menu_texture;
gef::Sprite menu_sprite;

StartState();
~StartState();
void Init(gef::Platform* platform_);
bool Update(float frame_time, gef::InputManager* input_manager_, ARApp* main_app);
void Render(gef::SpriteRenderer* sprite_renderer_, gef::Platform* platform_);
gef::Texture* CreateTextureFromPNG(const char* png_filename, gef::Platform& platform);
```

```
InstructionState

gef::Texture* menu_texture;
gef::Sprite menu_sprite;

InstructionState();
~InstructionState();
void Init(gef::Platform* platform_);
bool Update(float frame_time, gef::InputManager* input_manager_, ARApp* main_app);
void Render(gef::SpriteRenderer* sprite_renderer_, gef::Platform* platform_);
gef::Texture* CreateTextureFromPNG(const char* png_filename, gef::Platform& platform);
```

```
EndState

gef::Texture* screen_texture;
gef::Sprite screen_sprite;

EndState();
~EndState();
void Init(gef::Platform* platform_);
bool Update(float frame_time, gef::InputManager* input_manager_, ARApp* main_app);
void Render(gef::SpriteRenderer* sprite_renderer_, gef::Platform* platform_);
gef::Texture* CreateTextureFromPNG(const char* png_filename, gef::Platform& platform);
```

```
PlayState

float fps_;
Marker marker[3];
Marker centre;
GameObject beam_cube[3];
GameObject tower[3];
gef::Matrix44 mesh_matrix;
gef::Vector4 direction[3][3];
float dist;
float v_dist;
float h_dist;
float angle;
float timer;
int score;
std::vector<Character> character;
gef::Vector4 character_scale;
collision collision_manager;
gef::Scene* tower_scene_;
gef::Scene* bomb_scene_;
class gef::Mesh* tower_mesh_;
class gef::Mesh* bomb_mesh_;
gef::Texture* blue_bomb_tex;
gef::Texture* red_bomb_tex;
gef::Texture* tower_tex;
gef::Texture* lightning_tex;
gef::Material bomb_mat;
gef::Material tower_mat;
gef::Material lightning_mat;

PlayState();
~PlayState();
void Init(PrimitiveBuilder* primitive_builder_, gef::Platform* platform_);
bool Update(float frame_time, gef::InputManager* input_manager_, ARApp* main_app);
void Render(gef::Matrix44 proj_matrix, gef::Matrix44 view_matrix, gef::SpriteRenderer* sprite_renderer_,
gef::TextureVita& tex_vita, gef::Renderer3D* renderer_3d_, float cam_aspect_ratio, float display_aspect_ratio,
float vert_scale_factor, gef::Sprite cam_sprite, gef::Font* font_, gef::Platform& platform_);
void RenderOverlay(gef::SpriteRenderer* sprite_renderer_, gef::Platform& platform_, gef::Font* font_);
void DrawHUD(gef::Font* font_, gef::SpriteRenderer* sprite_renderer_);
Character SpawnCharacter();
gef::Mesh * GetFirstMesh(gef::Scene * scene, gef::Platform* platform_);
gef::Texture* CreateTextureFromPNG(const char* png_filename, gef::Platform& platform);
```

```
Collision

collision();
~collision();
bool AABBtoAABBCollision(gef::MeshInstance*, gef::MeshInstance*);
bool RayTriangleCollision(gef::Vector4 ray_origin, gef::Vector4 trangle_point[3]);
```

```
Marker

gef::Matrix44 transform;
gef::Matrix44 local_transform;
bool is_found;

Marker();
~Marker();
void SetTransform(gef::Matrix44 new_transform);
gef::Matrix44 GetTransform();
void SetLocalTransform(gef::Matrix44 origin_transform);
gef::Matrix44 GetLocalTransform();
void SetFound(bool found);
bool GetFound();
```

```
GameObject : MeshInstance

float rotation_x;
float rotation_y;
float rotation_z;

GameObject();
~GameObject();
gef::Vector4 Get_Position();
void Set_Position(gef::Vector4 new_position);
void Set_Rotation(float rot_x, float rot_y, float rot_z);
gef::Vector4 Get_Scale();
void Set_Scale(gef::Vector4 new_scale);
void set_transform(gef::Matrix44 new_transform);
void Build_Transform();
```

```
Character : Gameobject

gef::Vector4 direction;
float point_value;
gef::Texture* character_texture;

Character();
~Character();
gef::Vector4 GetDirection();
void SetDirection(gef::Vector4 new_direction);
float GetPoints();
void SetPoints(float new_value);
gef::Texture* GetTex();
void SetTex(gef::Texture* new_texture);
```

Problems and Solutions

PSVita Issues

One persistent problem was the use of the PSVita itself as the only way for work on the application to be completed was to use the PSVitas in the university. These are only available in one room which is often used for other classes and therefore work on the project had to be undertaken at specific times, limiting the amount of work that could be completed. The PSVitas would also sometimes not function perfectly and could disconnect, meaning that work had to be stopped in order to get it working again or to log onto another computer. This would break up the flow of work and be a distraction. Finally the PSVitas in the university must be plugged into the computers and are tethered by a metal cable which is understandable however the restricted movement sometimes made it harder to test the application with multiple markers at once. There was no way to prevent these issues however none of them were serious enough to cause a major issue. The time allocated to complete the project was used wisely in order to make the most of the time the PSVitas were available and so this was enough to prevent these issues from stopping all progress in the project.

Framework Unfamiliarity

As the framework was unfamiliar it was difficult to write code in a way that worked with the framework. This cause several problems to take much longer to be resolved as the solution was not immediately obvious. This was fixed by simply using the framework enough to help become familiar with it and then the issues that this unfamiliarity caused were easier to fix.

Off-Screen Marker

When a marker went off-screen the application would originally not know how to handle this and draw the objects in the wrong place. This was because it could not update its position relative to the screen and so the marker position was thought to still be on the same part of the PSVita screen. This was solved by interpolating the position of an invisible marker from the positions of visible markers and then updating it once again when the marker became visible again. This works well however when the markers are moved whilst off screen the application will be unable to update their position as there is no way for the system to gather information from something that is unseen by its camera.

Rotating Beams

One issue found whilst implementing the beam cubes was with the rotation of the cubes in order to have them stretched in the correct direction. What happened was that the beam would be placed correctly when the PSVita was held upright but when the PSVita was rotated, the beam was also rotate. This happened because the rotation of the AR marker was being taken into account when it was in fact irrelevant and so when the camera was rotated, the AR marker appeared to be at a different angle and therefore the beam would rotate further than desired. This was fixed by setting the rotation of the marker matrix to the identity matrix before multiplying it by the offset matrix, which took into account only the rotation of the offset matrix.

State Classes

Several issues were discovered when implementing different classes for the different game states. The first issue was that the classes for the main AR app and the individual states were not recognising each other despite including their header files. This was because classes are not able to have cyclical includes, meaning that since that AR app class included the state classes header file, the state class cannot include the AR app header file. This was fixed by forward referencing the AR app class in the state class header file and including the AR app header file in the state class application file. The second issue came when the application was running and returned from the render function in the play class. The function would run through without issue but when the back in the AR app class the vita texture would cause the application to crash. This was due to the fact that the vita texture was being modified in the play class render function but the texture was being passed to the class by value, and so those modifications were not taking effect in the AR app class. Changing the method of passing the variable from by value to by reference fixed this issue.

Loading Screen

The loading screen caused several issues as it was being developed. Firstly, after a loading screen image was created it was saved in the same location as every other saved image resource however when loaded into the game it did not render correctly. Other images were tested such as the textures of the objects which worked as expected but the newly made images did not. It was discovered that this was to do with the bit rate of the image and was fixed by opening the image in Microsoft paint (Microsoft, 1985) and re-exporting it. Another issue was that the screen was not rendered until the render phase was complete and so if the game started loading before this phase was completed once, the loading screen would not be displayed as the game loads, which is its sole purpose. To combat this a Boolean was created in the loading screen class which is set to false by default and gets changed to true when the screen has been rendered. When this Boolean is true, the update function initialises the game state and changes the state to the start menu.

Project Evaluation

The project has been very successful overall. The game is designed with the use of augmented reality markers in mind and therefore is easily picked up and played by those who do not play games regularly or are a young age. The game was also designed with simplicity in mind to appeal to this target audience and minimise issues in implementing features. The objects are rendered correctly in relation to the markers and look as they should from any angle and when the markers cannot be located the positions are correctly interpolated to ensure the objects still render as intended. The code is separated so that each game state is in a unique class, each containing their own initialise, update and render functions as well as any other functions that that particular class may require. The main AR app function is therefore left for creating variables for renderers and mesh builders that are required in the other classes, managing and changing the current game state and calling the relevant functions of the state classes.

There are aspects of the project that could be improved or expanded upon and if the project was to be continued beyond this coursework these are some of the changes that would be made.

- The central marker may be unnecessary if room tracking was used. The PSVita would be able to create a central transform by tracking the edges of shapes on the play surface. This would mean that the central marker could not be moved by the player and the game would require less equipment to play.
- The game, once completed, shows the score the player was able to achieve and allows them to quit to the main menu to play again, however a feature that would improve the gameplay somewhat is a leader board system so that the user can compare their scores to scores they have achieved in the past. This could be achieved by creating a text document of the previous scores and saving it to the PSVita memory. The list would be sorted to display the highest to lowest score and scores below the ten highest would be deleted from the document. This was not implemented as it does not help to demonstrate the use of AR

technology and so gameplay elements that were necessary to show this technology were prioritised.
- A pause menu that stops the game and allows the user to immediately return to the main menu was thought about during development however it was a low priority feature and time ran out for it to be implemented. Because of this the player will either need to wait for the timer to expire or quit using the PSVita home button in order to exit the game. This is not a major issue as the timer is only active for sixty seconds and there is not much reason to quit to the menu as there is only one game mode.

Innovation

The main opportunity given by the use of AR technology was simply the ability to render objects within an image of the real world. Because of this, considerations had to be made about where objects were rendered as whenever the PSVita was moved the objects would have to move to the respective position on-screen. This was a challenge, however the result allowed for objects to be realistically drawn over the real-time video feed, which is an interesting effect.

Another opportunity that would have been impossible without the use of AR technology was that the electrical towers could be moved around very easily by simply moving the markers in real life. This could have been done in a traditional video game by allowing the user to select a tower and then use an analogue stick or directional pad to move it around however this would be far more complex for the user. The towers could also be moved with the mouse or a finger on a touch screen however this would only allow the user to move the towers in two dimensions, whereas the markers can be easily moved around in three dimensions. If this was played on a touch screen another issue is that the user’s finger would likely obstruct their view of the game which can be avoided by using the markers.

References

Nintendo (2011) AR Games: Augmented Reality – Nintendo 3DS [Video game]. Nintendo.

Niantic (2016) Pokémon GO – iOS, Android [Video game]. Niantic.

SCE London Studio, Playlogic Game Factory (2009) EyePet – Playstation 3 [Video game]. SCE London Studio, Playlogic Game Factory.

TurboSquid (2000) Available at: https://www.turbosquid.com/ (Accessed: 21 November 2018).

dotPDN, LLC (2004) Paint.net [Computer program]. Available at: https://www.getpaint.net/download.html (Accessed: 10 October 2018).

Bensound Available at: https://www.bensound.com/ (Accessed: 25 November 2018).

Freesound (2005) Available at: https://freesound.org/ (Accessed: 25 November 2018).

Möller, Tomas; Trumbore, Ben (1997). "Fast, Minimum Storage Ray-Triangle Intersection". *Journal of Graphics Tools*. **2**: 21–28. doi:10.1080/10867651.1997.10487468.

Microsoft (1985) Paint [Computer program]. (Accessed: 20 November 2018).