1 Introduction

This application has been designed and developed with augmented reality in mind. The application is a game in which the user has to first locate an augmented reality marker, where a cube will appear floating above the marker. On this cube there are five different puzzles, one for each face except the base. The user then has to rotate the tiles till a path is from the star tile, the green tiles, to all the end tiles, blue tiles, has been completed. Once each face puzzle is done the face disappears and the user has to move onto the next face. Once all faces on a cube has been completed the user has to find another marker, until all size markers have been found. As well as using the camera of the PS Vita for the augmented reality aspect of the game the application utilise the front touch screen panel for on screen buttons.

2 User Guide:

After starting the application either press the physical START button or press the on screen start button. Now face the PS Vita toward an AR marker, once the marker is seen move the left analogue stick left and right to select the right cube to solve. Next move the left analogue stick up and down to choose the desired face. Once the correct cube and face is selected use the left and right shoulder buttons will rotate the selected tile left and right respectively. To change the selected tile press the D-Pad and the selected tile will move in the same direction.

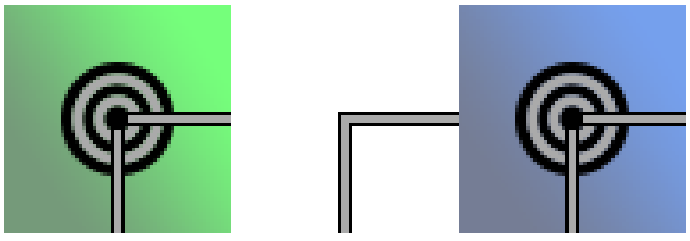

Figure 2.1: Puzzle Tiles: Left - Start, Centre - Normal, Right - End

To complete a puzzle rotate each of the tiles until there is a path from the start tile, as shown in Figure 2.1 as the tile with the green background, to all the end tiles, shown with a blue background. The centre tile are plain tiles that are used

as connectors. The tiles shown are just one variation of each of the three types of tiles. Once a puzzle is completed press either the on screen solve button or the SELECT button, this will check to see if there is a path between the start tile and all the end tiles. If there is then the face will disappear and the next face will be selected.

Once a cube is completed find another marker and select that cube to be solved. When every cube has been completed the game is over. The goal is to complete the game as quickly as possible whilst using the smallest number of solves possible. While playing the game pressing the on screen pause button or the START button will pause the game.

3 Design:

3.1 Technology:

When creating the application it was designed and built upon the main aspects of augmented reality applications, the use of a marker as an anchor in the 3D world. This application builds all the objects around a designated marker. Figure 3.1 shows the pseudo code that does this process.

```
for num of markers{
        if seen{
                set cube ar matrix();
        }else{
                clear ar matrix;
        }
}

set cube ar matrix(){
        for num of faces{
                set puzzle ar matrix();
        }
}

set puzzle ar matrix(){
        set ar matrix;
        build new world transform matrix;
}
```

Figure 3.1: Pseudo Code

In Figure 3.1 the augmented reality marker location matrix is sent to the main

cube object. From here there matrix is then passed to the faces of the cube to be used by the object as part of its own world matrix.

```
for each tile{
        //local rotation multiplied by local position for the puzzle
        build local martrix();
        //get the tile matrix and multiply by local matrix;
        tile matrix * local matrix;
        current matrix * ar matrix;
        set the world matrix of each tile to be the current matrix;
}
```

Figure 3.2: Matrix Pseudo Code

An example of how the world matrix is used by a tile in a puzzle can be seen in Figure 3.2. This code shows that each tile starts by first creating its own world matrix based upon its own rotation and position. Next this matrix is the multiplied by the matrix of the cube object it is relative to. This results in the position of the tile in the world being placed just above one of the faces of the cube. The next step is what sets the tile relative to the augmented reality marker, which is by multiplying the current matrix by the augmented reality matrix returned from the augmented reality marker.

As well as the application development working well with the exploits of augmented reality the gameplay of the application works well with how augmented reality works. As mentioned previously when playing the game the user has to solve the puzzle on each face. The best way to do this is by moving around the cube while changing the face needing to be solved at the time. This process allows the user to see each face of the cube without interrupting the gameplay to move the marker, which may not be possible if the marker is in a fixed location.

3.2 Software:

The application has been written as mentioned previously with the augmented reality technology in mind. Due to this the application has been built with object-oriented approach. This can be seen in how the objects are created. To start with there is a base game object which has variables and functions that are required by all objects in the application.

This class is then inherited from for the marker object which is designed around being relative to an augmented reality marker. This object has functions needed to do this such as setting an augmented reality matrix and building the world matrix based upon this new matrix. This is the type of the objects used to cover the augmented reality markers in the game.

The cube object and the puzzle objects do not inherit from these class but they do instantiate other classes. The cube object class instantiate both the puzzle object and the marker object classes. While the puzzle object class instantiates a game object. All this can be seen in Figure 3.2 with the blue arrows representing inheritance and red arrow representing instantiation. Another class that inherits is the augmented reality camera class as it adds functionality to the camera class that is specific to augmented reality applications.

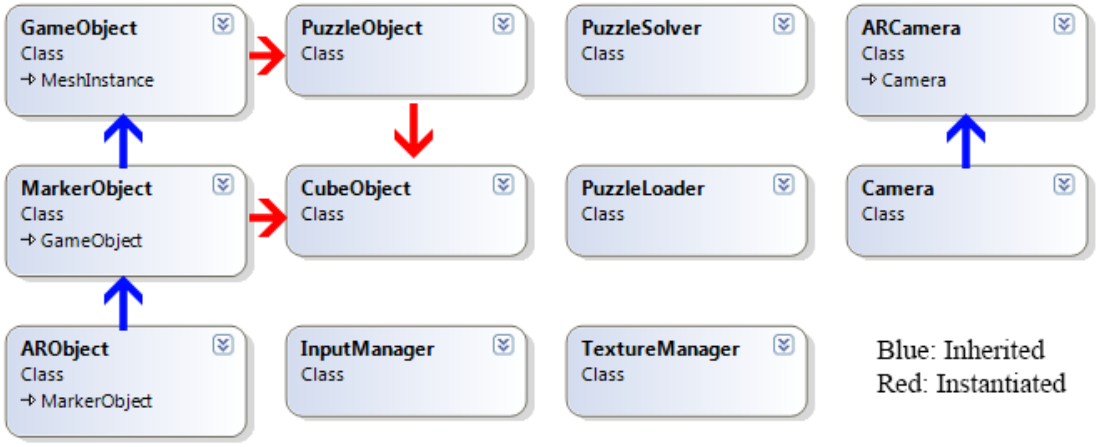

Figure 3.3: Class Diagram

As well as using object-oriented approach to the objects used in the application is a Singleton pattern which is used for a number of the classes that manage particular objects or only need to be created once. Currently in Figure 3.2 the class that use the singleton pattern are Texture Manager, Input Manager, and the Puzzle Loader and Solver classes. These classes use the pseudo code shown in Figure 3.4 to create only one instance of itself in addition with the constructor being a private function. This code is based upon an example provided by CodeProject (2002). As mentioned previously the reason for using this pattern is it allows for the classes

to be created once it also allows for the class to be used in different objects whilst being only initialised once, which is very useful for the Texture Manager.

```
ObjectType* GetObject(){
        if no instance of ObjectType{
                Create new ObjectType;
                set instance flag;
        }else{
                return already created ObjectType;
        }
}
```

Figure 3.4: Singleton Pseudo Code

Figure 3.5 shows pseudo code for the process used to check that the user has solved the puzzle. The application checks by following the path generated by the tiles. For the first run the location of the tile is the location of the start tile. The algorithm checks to see if the current tile is an end tile, which will be removed from the end tile list if it is. If there are no more end tiles then the puzzle has been completed and the loop is exited. Next each direction is check to see if it can move in that direction without leaving the bounds of the puzzle or entering a tile which isn't connected. If it finds a direction store it and check the other directions. If another direction has been found then add the tile to a list to be jumped back to later if the path hits a dead end. Once all directions have been check move to the next location and run again. This process is repeated until all the ends are found or there are no more tiles to move to.

This process is run every time the user either presses the on screen solver button or the Select button. The reason for waiting for user input rather than checking constantly or after every tile change is doing it that would take use a lot of resources that are not needed. Also waiting for user input stops the user from accidental solving the puzzle.

```
Set current location to be that of the start tile
while(running){
        if(end tile){
                remove from the end list
        }
        if(end list empty){
                solved puzzle;
        }
        if(direction clear){
                if(no new direction){
                        set new direction
                }else{
                        add tile to list if haven't already been added
                }
        }
        move to the new tile based on direction found
        if(no new direction){
                check tile list
                if(empty){
                        no solution and exit loop
                }else {
                        jump to new tile
                }
        }
}
```

Figure 3.5: Puzzle Solver Pseudo Code

4 Reflection:

4.1 Review:

4.2 Innovation:

References

CodeProject. 2002. *Singleton pattern and its implementation with c++*. [Online]. Available from: http://www.codeproject.com/Articles/1921/Singleton-Pattern-its-implementation-with-C. 3.2