

The Design & Development of RPS-Vita

An Augmented Reality Game for PlayStation Vita

CMP404.2016-7.S1: Applied Game Technology

Duncan Bunting | 1302739

1 - Design

1.1 - About The Game

“RPS-Vita”, or “Rock Paper Scissors - Vita” is an augmented reality game based on an old Dexter’s Laboratory (Dexter’s Laboratory 1996) game “Bot Brigade” (Dexter’s Laboratory - Bot Brigade [no date]) a flash game was featured on Cartoon Network’s website (Cartoon Network 1992). In this game a player is pitted against an AI, each being able to spawn three units that will travel to the opponent's side of the arena in an attempt to hit their base which will win the round. As with the Dexter’s Laboratory game this game uses the core simple mechanics of Rock, Paper, Scissors while adding a few extra features to produce more exciting gameplay including moving barriers protecting each base and different game modes to alter difficulty.

1.2 - Rock, Paper, Scissors

Following the well known hand-game “Rock, Paper, Scissors”, the game allows you to pick between three units to spawn each with their own strength and weakness. It is up to the player to determine the best unit to spawn and when to spawn it. If two opposing units meet, the winner continues on towards the opposite side of the arena while the other is destroyed. Each unit can be identified by their unique models, and players may only summon 3 units at a time with a small delay between being able to spawn a new unit.

1.3 - Lanes

Similar to the Dexter’s Laboratory game that featured seven lanes, RPS-Vita features three. This choice of gameplay mechanic allows for a more interesting combination of strategy and risk, while a player can spawn a unit on a lane that is empty they must also keep in mind that at the same time the opponent is beating them in another lane.

1.4 - Barriers

Each player has two sets of barriers, each set containing 4 barriers and moving in opposite directions to each other, creating a field of protection against enemy units approaching the base. These barriers move at a faster speed to units and the key to getting a better score is finding the quickest way past them. Once a unit collides with a barrier, both the barrier and unit are destroyed leaving a gap in the barriers.

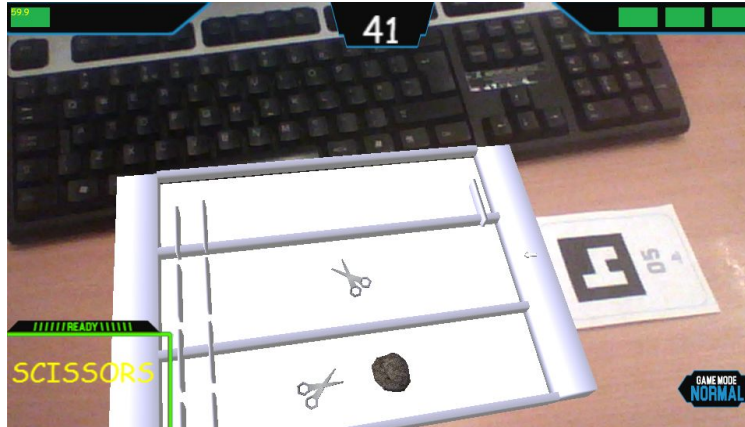


Figure 1 - *Gameplay Screenshot*

1.5 - HUD

The game features a simple heads up display, which displays each player's health, the currently selected unit and the game-time. In addition to the basic HUD, several pop-up sprites can appear under certain conditions, such as, if the main marker is lost a warning message comes up telling the player to relocate the marker, see Figure 2. In the viewer, a window with information on the model being currently viewed will appear and if the game mode is changed, or if a player wins or loses appropriate pop-ups appear.

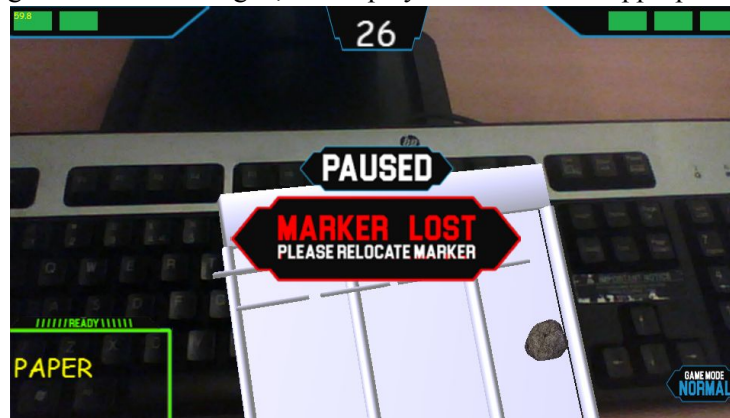


Figure 2 - *Game Screenshot, showing Marker Lost message.*

1.6 - Augmented Reality

Augmented Reality plays a large role in the game, it is a strong personal belief that no one gameplay element should feature augmented reality but instead have the world the game is played in be in AR, this lead me to the decision of creating the entire game in augmented reality, including the main menu. The game does not simply have augmented reality as a feature but the use of augmented reality enhances the gameplay and merges a game for the PS-Vita with card games. In RPS-Vita, the player plays on two fronts, the augmented reality view of the events unfolding in the game on the PS-Vita and the cards laid out in front of them.

The game is played on Card 6, this acts as the main card and is required to play the game. This card places the main menu and the arena that the game is played in. If during the game, sight of this card is lost, the game is paused to allow the player to relocate the marker, ensuring that they do not lose due to a marker not being properly detected which is often the biggest frustration when using AR markers.

To spawn a unit, the player must select a card 3, 4 or 5 - Rock, Paper or Scissors respectively - placing the below the arena marker, the position of the card determines which lane the unit will spawn in, directly

below the arena marker is centre lane, to the left of that is left lane and to the right is the right lane - this is visually represented by a cursor that appears above the player's base pointing down the selected lane. Pressing X on the PS-Vita confirms the selection and spawns the unit.

1.7 - Model Viewer






A simple addition to the game is a viewer which allows you to view any of the 3 unit models as well as information about each unit. The purpose is to show off the models for each unit in a larger scale than on the arena while also giving new players an insight into what beats what. The viewer can be accessed through the main menu. Figure 3 shows when the marker belonging to “Scissors” is placed in-front of the camera.








Figure 3 - Model Viewer, showing Scissors.

2 - How-To-Play

2.1 - Controls / Markers

Anywhere	
	Recalibrate White Balance
	Return to Previous State (ie: Viewer to Menu)
Menu	
	Confirm Selection
	Change Selection
	Display's Menu

Game	
	Spawn Unit
 01	Changes Game-Mode to Double Speed
 02	Changes Game-Mode to Half Speed
 03 04 05	Selects “Rock”, “Paper”, “Scissors” respectively when in view. - Place to the left, right or in-line of Marker 6 to select which lane to spawn in. (See Figure 1)
 06	Places Arena

2.2 - Guide

It is recommended to play the game with your right hand holding the Vita at all times and to leave your left hand free to manage the AR cards. Start by placing Marker 6 on a flat surface and facing the PS-Vita’s camera towards it, this marker will start by being the main menu but will later be where the game’s arena is placed. In the main menu use the arrow-keys to navigate between the 4 options. Choose an option by pressing Cross.

Choosing How-To-Play will display a image on-screen with a short description on how to play the game. This can be closed by press Cross again or Circle.

Choosing Viewer will take you to the model viewer, this allows you to place Marker’s 3, 4 or 5 in-front of the Vita to display the model of each “monster” and information about them including their weaknesses.

Choosing Play will begin the game. The game will start immediately after selecting play. To select a unit to spawn, select from Markers 3, 4 or 5 again and place them either to the left, right or in-line with the arena marker to determine the lane to spawn a unit in - this is visually represented by the arrow model sitting above your base and which unit will spawn is shown by text to the bottom left of the screen. To confirm and spawn your unit press Cross. Your unit will progress down the arena until it either reaches the enemy’s base, barriers or units. The victor of colliding with an opposing unit is determined by the

classic rules of “Rock, Paper, Scissors”, if colliding with a barrier both the barrier and unit are destroyed and if colliding the the enemy base the round will end and your opponent will lose a life, of-which both you and the enemy have 3 of and is displayed at the top of the screen.

The difficulty/gamemode can be changed by placing Marker 1 or 2 in view of the Vita’s camera. Marker 1 will make the game faster, Marker 2 will slow it down and if neither are present, the game will go back to its regular speed.

Play until either the AI beats you or you beat the AI. Score is judged by how long it takes you to beat the AI; a lower score is better!

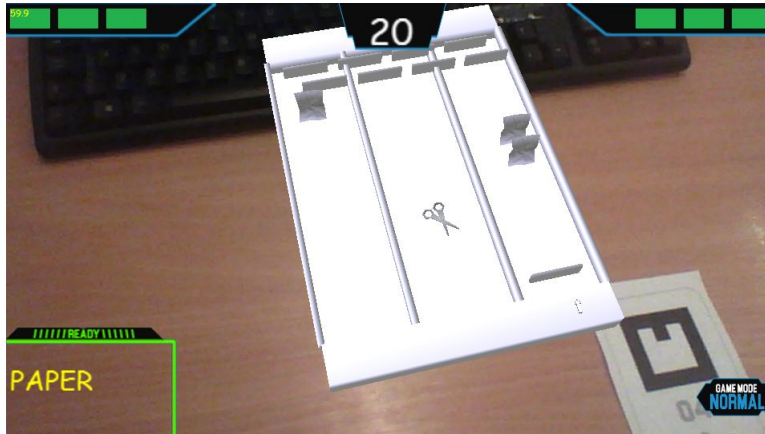


Figure 4 - *Gameplay Screenshot, showing spawning Paper on Right lane.*

3. Development

3.1 - GameObject Class

Control over all game objects in RPS-Vita is achieved through the Game Object class, this class includes a series of getters and setters including the standard: Position, Velocity, Rotation, Scale. In addition to these I have added a few extra to assist the application:

Active - Boolean, Controls whether the object should be drawn or not.

SetMesh - Allows the changing of the currently set model.

Collidable - Boolean, Whether or not the object should trigger collision messages.

GameState - Taken from the public enum within the Game class. Sets the state that the object will appear in.

All of these functions proved incredibly useful throughout the programming of the application and allows the easy organization of game objects according to the attributes given to them.

Game Objects are implemented through an array in `agt_cw.h/cpp`, although not all are used there are “static const int” variables for each game object, allowing to easily edit specific objects without magic numbers.

3.2 - Other Classes

In addition to the GameObject class, the application features several other classes to allow for neat and easy access to commonly used code. The CollisionClass, which simply allows easier readability changing the required code to detected collision from:

```
CollisionAABB(player_.mesh()->aabb().Transform(player_.transform()),  
player2_.mesh()->aabb().Transform(player2_.transform()))
```

to

```
collision_->CollisionAABB(gameObject[i], gameObject[j])
```

The PlayerClass, stores information about the two players (Human and AI) such as the health and which lane and unit is currently selected.

The SpawnerClass manages the spawning of units, allowing for a neat and organized way of detecting which marker is being detected and in what position the marker is in to spawn the correct unit in the correct lane.

The SpriteClass, is very similar to the GameObject class but instead of handing mesh's it loads textures from .PNG files and stores the appropriate information with access to anything that might require changing via getters & setters.

3.3 - Game Class & Game States

The final class that is made use of is the Game Class; storing all the vital information about the game, again to allow it to be easily and neatly accessed. This also allows the easy reset of any variables that determine starting a new game. The most vital part of the Game Class is the publicly declared enum GameState; this enum allows the management of which state the game is currently in, from: Menu, Viewer and Game. The GameState is vital in determining what objects and sprites are to be drawn; as previously mentioned, sprites and GameObjects are drawn depending on the game state they belong to.

3.4 - Marker Detection & Interaction

RPS-Vita makes use of all the provided standard PS-Vita AR cards. All markers have a unique function and the main marker, 6, acts as the main marker, drawing both the Menu (which is drawn in AR) and the arena that the game is played in - this is managed thanks to the previously mentioned GameState from the Game Class. For the main game, marker 6 must be detected for the game to play, if it is not detected a sprite is activated prompting the user to relocate the marker. After establishing the position of marker 6, the game has to detect which marker is present to spawn a unit and where to spawn it. This is achieved by finding the spawn marker's (either 3, 4 or 5) position in relation to the main marker; if the spawn marker is below the main marker, the unit is spawned in the centre lane, if to the left of the main marker, it will spawn to the left and to the right for right of the marker. findMarker() detects markers in order using a standard if-elseif statement, this means that if more than one spawn marker is present then it will take the lowest number first (ie. Rock & Scissors markers are present, it will select rock). To avoid confusion on what will be spawning, text is displayed on screen to confirm the currently selected unit. The selected lane is also shown via an arrow that can be seen in the AR.

3.5 - Problem Areas

3.5.1.1 - Lots of Game Objects

The first problem encountered was realising the amount of gameobjects and their different functions and attributes. I found myself writing a very large amount of code for things that should have been done in a couple lines if not one. In addition, there were a number of attributes that I wanted to keep track of such as if the object should be drawn and if it was collidable.

3.5.1.2 - Solution

Thankfully, this problem was easily solved through the early creation of the GameObject class and the use of getters and setters, drastically reducing the amount of code required. To solve the problem with the sheer amount of gameobjects that would be present, all game objects were created using an array and although not all are referred to individually, “static const int” variables store the ID’s to all the game objects allowing them to be accessed easily without the use of magic numbers, ie: `gameObject[GO_UE3].Active()` will return whether or not the game object for “Enemy Unit 3” is active or not.

3.5.2.1 - Losing the Marker & Game Is Too Hard

During testing of the game throughout the development, a continuous frustration was the vita not detecting the main marker for one reason or another. Originally the game would continue to play and update even if you could not see the marker; overall this made the user experience much worse, forcing users to have the marker in sight as soon as the game started and not being able to move around too much risking losing detection of the marker. In addition to this, during early play-tests users were not able to keep up with the game and found it incredibly difficult to get a grasp of; however, once established (after a sizable number of attempts) they were able to easily beat the “AI” and some said the game got too easy.

3.5.2.2 - Solution

To solve both problems I introduced the variable “_speed” in the Game Class, this would control the speed the game is played. Movement of all moving objects is manipulated by `game.Speed()` allowing me to not only stop objects from moving when the main marker is lost by setting `game.Speed(0.0f)` but also me to change the difficulty of the game by adjusting the speed variable; this prompted the introduction of a new mechanic to the game using marker cards 1 and 2, which start Speed and Slow mode respectively - with neither marker being present returning to normal speed. Slow mode has received positive feedback allowing users to get used to the game before returning to normal speed and advanced users having the option of making it very hard by increasing the speed.

3.5.3.1 - Controlling Markers & The Vita

Once gameplay mechanics were established and the game became a properly playable game, I both observed play-testers and myself having extreme difficulty being able to control the game. Originally, the game would require you to select your lane with the arrow buttons on the PS-Vita and press X to confirm; all while holding the Vita and managing the AR cards to make sure only the one you wanted was visible to the Vita.

3.5.3.2 - Solution

To overcome this problem and to further the quality of the application, use of the arrow buttons were removed (except for the Menu) and instead of selecting which lane to spawn in on the Vita I included that in the marker detection. I created a class to manage if a spawn marker was detected ("spawner.h"/cpp) and included the following code:

```
if (!spawner.Detected())
{
    if (sampleIsMarkerFound(i) && game.State() == GS_Game)
    {
        inv_markerTransform.AffineInverse(markerTransform);
        gef::Matrix44 spawnMarker_Transform, spawnMarker_localTransform;
        sampleGetTransform(i, &spawnMarker_Transform);

        spawnMarker_localTransform = spawnMarker_Transform*inv_markerTransform;

        spawner.Update(i, spawnMarker_localTransform.GetTranslation().x());
    }
}
```

This code calculates the spawn marker position in relation to the main marker's position (markerTransform), this allows me to see whether the spawn marker is to the left, right or in-line with the main marker, allowing the user to select the lane in relation to where there card is - effectively taking the game from being a Vita game with AR, to an AR & Card game. The user now plays on two fronts, visually able to see the game on the vita while effectively playing a card game on the surface in front of them at the same time - it also allows the vita to be held with the right hand, and card managed with left hand.

3.5.4 - Unfixed Bug: Menu Disappears

There is one unfixed bug that remains in the game. Because the conditions that cause this are currently unknown, occasionally the when going back to the main menu the main menu game object will not draw. The game object is active and is being called to draw, but for some reason is not spawning. I have attempted to fix the problem but due to limited time since when the bug was first noticed and it, appearing to be, a rare occurrence.

3.6 - Critical Analysis

Overall I am very proud of the work accomplished from the development of this game; however, it is not without faults. I feel I have got partially lost in the mindset of "the code doesn't matter, if it works it works.", a mindset I strongly disagree with but I still feel like some of my code lacks quality over the need to get it my target completed in time. Until very late into development I made little use of classes and code organization was unsatisfactory, it wasn't until the end was drawing close that I started to focus on cleaning the code. Because it was left so late there are still some areas which could be greatly improved on but would require more time than was available. For example, although I am happy that I have managed game objects relatively well and what happens during each game state, I feel this code is spread out too much and could have been condensed and cleaned up more, for starters, the first change I would make would be drastically reducing the size of the findMarker() and Input functions. As I feel there is sizable important chunks of code that should not be present in either and instead should be moved to Update() or their own functions; for example, rather than having all the code in findMarker() set a boolean for each marker and have functions called in Update() depending on what markers are called. This would greatly improve readability of the code. Some other areas such as the use of arrays I feel are not optimal

and if I had done more research on better ways to approach problems the quality of code could be improved greatly, instead I stuck to what I knew.

With that said, I am still extremely pleased with the outcome. The game has received extremely positive feedback and other classmates were assisted through seeing how I did certain things so overall I feel very proud of what I have managed to accomplish in the allocated time. In addition to this, the base of this game could very easily be converted and used to create other games with minimal changes outside of the main (agt_cw.h/cpp) files.

3.7 - Future Development

3.7.1 - How My Game Relates & Mixed Reality

I have very strong opinions about AR games, for the most part, being done incorrectly. Many games, such as Pokémon Go (Pokémon Go 2016) implement AR in one area of their game where it is not needed and what many believe to be better without AR and with many games calling a game that has a camera-feed as the background, AR. Because of this I went into this project knowing that I must create the game to properly merge reality and augmented reality, the end result is a game that acts as both a card game and Vita-game but neither work without the other. This is what I feel the correct direction of AR should be, merging the use of reality with the augmented reality. In addition, allowing the “arena” of a game to be built where the user declares, in augmented reality, allows for the game to be played in different environments and could possibly allow players to create their own level with real-world props. This would greatly assist in creation of new games for old genres and board games such as Dungeons & Dragons (Dungeons & Dragons 1974) where a story can be made up and maps physically created by players, the possibility of having that displayed in AR as you play is truly interesting.

3.7.2 - The Viewing Device / HoloLens

Although this application is not particularly hard to play, as it was designed for left hand on card and right hand on the PS-Vita, it still has the lead problem of holding a heavy PS-Vita and looking through its rather small video feed is quite an inconvenience. It appears the industry has realised that such small viewing areas that are given by the PS-Vita or even mobile phones is simply not enough for gaming. Microsoft (Microsoft Corporation 1975) I feel are leading the charge into AR both now and in the close future with their recently announced innovative product HoloLens (Microsoft Corporation [no date]), Figure 5 (Anon 2016).



Figure 5 - *The HoloLens* (Anon 2016)

This leap in AR hardware leads to many possibilities and will be exciting to see what developers make with the new technology. The main attraction being the complete vision of the augmented reality world, as well as it being hands-free, which could prove even more interesting with hardware such as the Leap Motion (Leap Motion, Inc. 2010) which can allow for motion tracking without the use of hand-held

trackers such as the PlayStation Move (Sony Computer Entertainment 2010). If my game were to be made for the HoloLens I have no doubt it would be a success if made into a multiplayer.

3.7.3 - VR vs AR

Currently both the industry and public are focusing mostly on the newly introduced Virtual Reality hardware like the HTC Vive (HTC Corporation 2016) which instead of mixing reality takes the user to a new virtual reality that, generally, does not interact with the real world. VR as a gaming platform, at the moment, seems a lot more appealing than AR simply due to the powerful graphics capable and not being limited by your surroundings; however, this shouldn't be a reason to push AR aside, AR opens to other possibilities and different games than VR offers. AR can innovate areas outside of games as well, manufacturers have started to gain interest and the possibilities of integrating AR with manufacturing and other professions opens a whole new area of innovation. The main advantage of AR over VR is not having any restriction on space, AR is augmenting the real world, you are able to travel anywhere and use it anywhere while VR currently requires you to be plugged into a computer and a set space to allow for movement, walking in VR is greatly restricted currently and AR smashes through this barrier.

3.7.4 - Conclusion

The possibilities of AR are definitely interesting and the way the gaming industry treats it may be completely different to its end practical use, with this said, the current progression in AR hardware supports my theory that the best kind of AR is properly mixing reality with virtual reality, cheap AR tactics that are featured in the majority of today's AR games for mobile devices does not demonstrate the true potential of the technology and it will be very interesting to see what game and software developers can achieve with the new upcoming hardware; however, for now I feel, VR delivers the best alternative reality experience for gaming.

4. References

- Cartoon Network. 1992. Available from: <http://www.cartoonnetwork.co.uk/> [Accessed 29 November 2016]
- Dexter's Laboratory*. 1996. [television programme]. Cartoon Network.
- Dexter's Laboratory - Bot Brigade*. [no date]. [computer game]. PC. Cartoon Network.
- Dungeons & Dragons*. 1974. [board game]. TSR, Inc.
- HTC Vive. 2016. [hardware]. HTC Corporation.
- Leap Motion Controller. 2010. [hardware]. Leap Motion, Inc.
- Microsoft Corporation. 1975. [corporation].
- Microsoft HoloLens. [no date]. [hardware]. Microsoft Corporation.
- PlayStation Move. 2010. [hardware]. Sony Computer Entertainment.
- Pokémon Go*. 2016. [computer game]. Android, iOS. Niantic.