

Giguesaur: Game Logic

Ashley Manson

September 28, 2015

Co-Workers: Joshua La Pine & Shahne Rodgers

Supervisors: Geoff Wyvill & David Eyers

Dept. of Computer Science
University of Otago

Contents

1	Introduction	2
1.1	Overview	2
1.2	Background	2
1.3	iOS and Android	3
1.4	Augmented Reality	3
1.5	Game Logic	4
2	Achievements	5
2.1	Prototype	5
2.2	Game Mechanics	6
2.2.1	Piece Rotations	6
2.2.2	Snapping Pieces	6
2.2.3	Picking Up and Placing Pieces	7
2.3	Port to iPad	7
2.4	Rendering	7
2.5	Integration	7
3	Results	8
4	Conclusion	8
4.1	Final Build	8
4.2	Future Work	8
4.3	Final Words	8
5	References	10

1 Introduction

Our vision for our completed Giguesaur application was allowing a classroom of children, each with their own iPad, to run around and solve a jigsaw puzzle together. Imagine a classroom full of kids where they are all trying to work on a single conventional jigsaw puzzle; such a scheme is in no way practical. The main goal of our project, besides all the design and technical subgoals, was simply to make a that is fun for children to play and work together.

1.1 Overview

The Giguesaur application development was divided into three different components. Joshua La Pine was in charge of developing the computer vision part of the project, which allows for the puzzle pieces to be rendered over top the ‘game board’ in the real world. Shahne Rodgers took charge of the networking component of the project, which was crucial in allowing more than one player to interact with the jigsaw puzzle. Finally my part of the project was to develop the game logic and render the game to the iPad’s screen.

1.2 Background

There are many jigsaw puzzle games that are available for iOS and other handheld devices, such as Magic Jigsaw Puzzles [1] and Jigsaw Puzzle [2], but the majority of them are limited in the way they look due to them using an orthographic projection to render the jigsaw puzzles. What this projection means is that the puzzle pieces of the jigsaw puzzle are flat on the screen, the player can only look at the jigsaw puzzle from top down, there is no depth and all the puzzle pieces are displayed as the same size. This is something that does not work for the Giguesaur project as the puzzle pieces are rendered in a perspective projection, so when the jigsaw puzzle is rendered onto the game board it looks more realistic, as puzzle pieces that are further away from the camera are shown to be smaller than pieces that are closer to the camera. Another limitation of jigsaw puzzle games is the way they can be interacted with, by which I mean the way they can be solved. The puzzle pieces have to be placed into a predefined grid. Farms And Animals Puzzles [3] is an example of a game that has this grid layout for the puzzle pieces to be placed in, which is shown in figure 1, it also shows the limited orthographic perspective of the game. The grid means that all the puzzle pieces will be placed in the centre of the screen. This is something that was illogical for the project and I did not want to limit the scope of the game. I have made it possible for the jigsaw puzzle to be solved anywhere on the board, be it in the centre of the game board or off in a corner of the game board. I also believe that it makes for a more interesting game.



Figure 1: Screenshot of Farms And Animals Puzzles [4].

1.3 iOS and Android

iOS is a phone operating system made for Apples small number of hand held devices, which are similar in many ways such as hardware and software implementaions. Android on the otherhand is an open source operating system based on linux of which is ran on a large number of devices crossing multiple different versions of the operating system. The devices that Android are ran on vary with a wide range of screen resoulutions and processing power. We had decided that the Gigusaur game would be developed for iOS for a couple of reasons. iOS hardware was more standardized than Android, as Apple are the only developers of iPhones and iPads it made it easy for us to write the code, knowing that we would be relativly safe that what we developed would work on the majority of iOS devices, at least the ones with cameras. We couldn't be certain that any Android development of an application would work, nor would it have been possible for a forth year project to ensure that the application would have worked on all kinds of different Android devices. Apple also have a powerful and intuitive integrated development environemnt (IDE) called XCode, that made development of the Giguesaur application very easy, with its detailed profiling tools and other features such as interface builders, it made it easier for us to develop and quickly prototype our ideas for the application.

1.4 Augmented Reality

Augmented reality is the idea of superimposing virtual objects on top of the real world in a realistic way that looks convincing enough for someone to believe

what they are seeing is actually physically real. In figure 2 it shows a game world being played on a simple coffee table. The game is a simple tower defence game where the user aims where the tower shoots by moving the iPhone around the table while the camera has a marker on the table in view. The marker on the table, of which is hidden under the projection of the defence tower, is what allows the game to get the required information to correctly pose the game objects onto the table. The idea of augmented reality is what we based the Giguesaur game on, using marker trackers to correctly pose the puzzle pieces onto the ‘game board’ so it gives the idea that we are interacting with jigsaw puzzle pieces in the real world.



Figure 2: Screenshot of ARDefender [5].

1.5 Game Logic

As I stated previously, I was in charge of developing the game logic for the Giguesaur game. This meant I had to create the logic for how jigsaw puzzle pieces interacted with each other and how the user interacted with the game. The jigsaw puzzle is made up of a grid with a specific number of rows and columns. Each piece has four edges, and an edge either has a neighbouring piece or not. An edge of a piece can be open, meaning it has not joined to its neighbour currently or closed meaning it has joined to its neighbour currently, or if the edge has no neighbour it is invalid, meaning it can't join or be joined to another piece for that edge. An invalid edge would be the outside edge of the puzzle. Each piece of the puzzle has a unique ID, a position in space, or an x, y coordinate on the board, and a rotation with a value between 0 and 360 degrees. The z coordinate is assumed to be 0, so it is ignored. This is how I developed the game logic. I made up my own data structure to store these details, the ID determines the index of the array of pieces where the piece is

held, the x, y coordinate determines where on the board the piece is displayed and the rotation affects the orientation of the piece. The board that the pieces are placed on has a width and length, which are along the x, y axis, which confines where the puzzle pieces can be placed, as all the pieces have an x, y coordinate.

2 Achievements

2.1 Prototype

In the beginning of the project I proceeded to develop a prototype of the Gigue-saur game on the Mac, using some simple OpenGL routines to render the game. Working on the Mac helped when I was creating the game logic for the application, as I could quickly test my ideas for the game logic such as jigsaw puzzle piece interaction. It also allowed me to quickly try out ideas that would be time consuming on an iPad. What I achieved with the prototype is the following; It allowed me to get to grips with OpenGL, such as calling all the drawing routines and getting to grips with the coordinate system, I completed and implemented all the game mechanics for the game, such as picking up and dropping pieces and the snapping of pieces, and I also tested out using a perspective projection to render the game in a proper perspective. Figure 3 shows what the game looks like with a perspective projection. The broken up picture of the puppy [6] represents the jigsaw puzzle pieces, the green background represents the game board the pieces are places on, and the white boarders on the edge of the game board is the ‘out of bounds’ area of the game board, where the pieces could not be placed. It shows off the mechanic of pieces being snapped together, as some of the jigsaw puzzle has been put together to be solved.



Figure 3: Screenshot of a prototype build on the Mac.

2.2 Game Mechanics

2.2.1 Piece Rotations

Each piece has an x, y coordinate location which defines where on the game board the pieces is rendered, they also have a rotation variable that defines how the piece is oriented on the game board. The rotation is around the z axis, where the z axis is pointing out from the screen. The addition of rotation for puzzle pieces added some complications to the logic of the game, in particular how I calculated the distance between pieces for snapping as well as rendering the pieces on screen with the correct rotations. Originally when I calculated the distance between pieces I had been doing a simple check to see if the pieces corner coordinates plus the length of a piece was in the range of the corner coordinates of its neighbour. This worked fine but failed to work when piece rotation were added, as adding the side length to the coordinates did not take into consideration the rotation. In a replacement to this, I know calculate the distance between piece corners using Euclidean distance, as with this I do not have to think about the rotations of the pieces as the calculation will return the correct values. To apply the rotation to the piece itself I use the following simple formula:

$$\begin{aligned}x' &= \cos(\theta) \times (x - x_cen) - \sin(\theta) \times (y - y_cen) + x_cen \\y' &= \sin(\theta) \times (x - x_cen) + \cos(\theta) \times (y - y_cen) + y_cen\end{aligned}$$

Where x and y are the piece corner coordinates, x_cen and y_cen are the piece centre coordinates, θ is the piece rotation in radians, and x' and y' are the new corner coordinates. I do this for all four corners of a piece to get the correct rotated coordinates of the piece.

2.2.2 Snapping Pieces

I wanted to enable a feature that allowed pieces to snap together, meaning if a piece was in a specified range to its neighbour, the piece being placed back on the board would move so that the two pieces were right next to each other, or they snapped together, which is shown in the Java Jigsaw Puzzle game made by Centurio [7]. This also has the added bonus of making it easier to check if pieces are right next to each other, rather than relying on the user to carefully place pieces together. How the check is made to see if two pieces should snap together is by comparing the distances between the corresponding corner points, and if the distances are less than the snap variable, the pieces snap together. So if the piece being put back on the board is being snapped to its left neighbour piece, the original piece top left point and the neighbours top right point would be checked, as well as the original piece bottom left point and the neighbour's bottom right point would be checked. The reason two distances are checked is to make sure that piece rotations are taken into consideration when snapping pieces together, so as to avoid an unusual snap where a piece rotated 90 degrees snaps to its neighbour not rotated. When a piece is snapped to another piece, the

piece saves the rotation of its neighbour as its own, so that when the change is rendered, they both have the same rotation when beside each other. For a puzzle to be considered solved, all the pieces should be snapped to their corresponding neighbours. Once a piece has snapped to its neighbour, a variable for that edge is set as closed, as well as the neighbour's edge. So if a piece has snapped to its right neighbour, its right edge is set as closed and the neighbour's left edge is set as closed. The check to see if the puzzle has been solved goes through all the pieces' edges to see if they are all closed.

2.2.3 Picking Up and Placing Pieces

To pick up a piece the user taps the image of the piece on the screen. That piece that the user has tapped is then stored in the users 'inventory' and they are no longer able to pick up another piece, until they have placed the piece that they are holding back onto the board. The inventory is just a term I am using to state if the user is holding or not holding a piece, meaning it can be easily checked to see if it is empty or not. The screen x, y coordinate from a finger tap are converted to a board x, y coordinate, z is ignored as it is assumed to be 0. If this is close enough to a piece then that piece is added to the user's inventory. This allows for the board to be rendered with a perspective projection, making the scene more realistic and achieving the augmented reality feel. When the user is holding a piece, it is moved to the iPad centre of the screen and projected onto the board, meaning the user has to rotate and move the iPad around to move the piece around the board. Once the user taps the screen again, where the piece is projected on the board is where the piece will be placed.

2.3 Port to iPad

Once I had established a working prototype for the Mac, the next step was to port the code to an iPad.

2.4 Rendering

The first major goal was to have a jigsaw puzzle rendered to the screen and the ability to move the jigsaw puzzle pieces around. I used OpenGL to do all the rendering of the jigsaw puzzle, using the coordinates of the jigsaw puzzle pieces, their current rotation and side length of the puzzle pieces as parameters to get them displayed correctly. With the parameters of each of the puzzle pieces, I get the four coordinates of their corners, with these I pass them to OpenGL and the puzzle pieces are rendered to the screen.

2.5 Integration

After I had successfully ported the application to the iPad and had everything from the prototype working correctly, we as a team had to work together on integrating all three parts of the project into a single coherent piece of software.

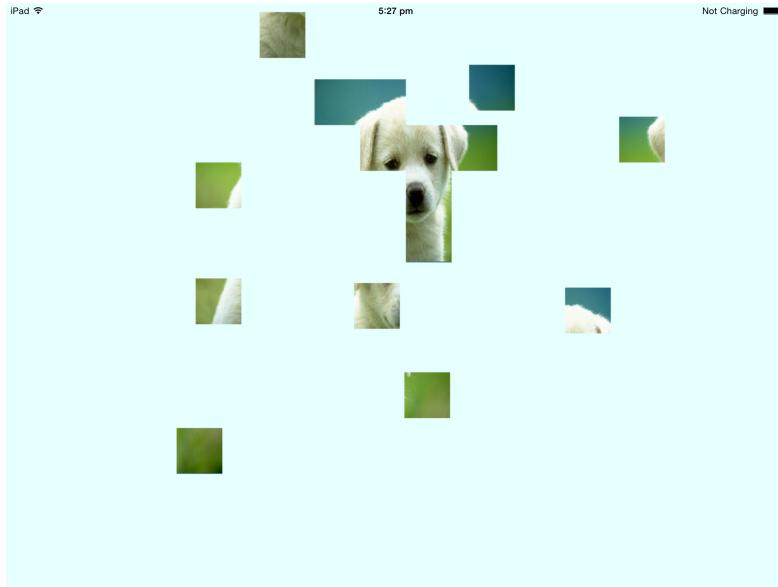


Figure 4: Screenshot of the first port to iPad.

The main issue we all quickly found out was that this task proved to be more difficult than previously thought.

3 Results

The finished product is a fully functional, interconnected, jigsaw puzzle solving game.

4 Conclusion

4.1 Final Build

4.2 Future Work

4.3 Final Words



Figure 5: Screenshot of the final build running on an iPad.

5 References

- [1] XIMAD, Inc, “Magic Jigsaw Puzzles.” <https://itunes.apple.com/nz/app/magic-jigsaw-puzzles/id439873467?mt=8>, 2015.
- [2] Critical Hit Software, LLC, “Jigsaw Puzzle.” <https://itunes.apple.com/nz/app/jigsaw-puzzle/id495583717?mt=8>, 2015.
- [3] D. Torbichuk, “Farms And Animals Puzzles - Jigsaw puzzle for children.” <https://itunes.apple.com/us/app/farm-animals-puzzles-jigsaw/id543511649?mt=8>, 2012.
- [4] AppColt, “Download Farm And Animals Puzzles - Jigsaw puzzle for children iPhone iPad iOS.” <http://img-ipad.lisisoft.com/img/2/7/2740-2-farm-animals-puzzles-jigsaw.jpg>, 2014. File: 2740-2-farm-animals-puzzles-jigsaw.jpg.
- [5] F. Quiquere, “ARDefender: iPhone Augmented Reality game by int13.” <https://www.youtube.com/watch?v=rB5xUStsUs4>, 2010. Screenshot at 0:26.
- [6] A. Overvoorde, “Textures objects and parameters.” <https://open.gl/content/code/sample2.png>, 2012. File: sample2.png.
- [7] Centurio, “Jigsaw Puzzle.” <http://sourceforge.net/projects/jigsawpuzzle/>, 2013. Sourceforge Source Code.