Quantum Hearts:
2D Puzzle Game for Android OS using OpenGL ES

Abraham Botros
SUNet ID: abotros
abotros@stanford.edu
Stanford University, Summer 2015 - CS148 Final Project Report

1 Introduction

"Quantum Hearts" is a simple 2D, flat-world, graphical puzzle game I created for Android OS using OpenGL ES (2.0).

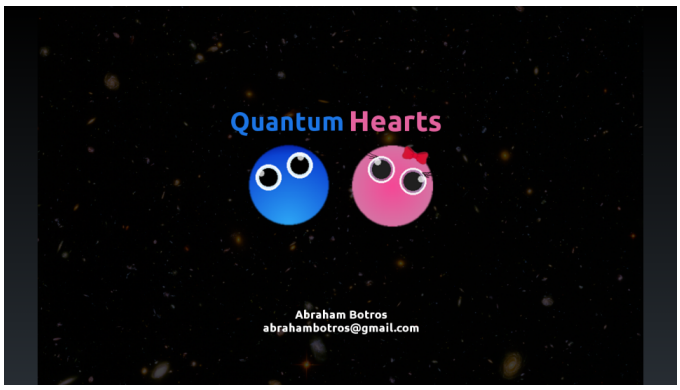


Figure 1: Title screen for the game, with Quincy on the left and Leah on the right.

It aims to tell the story of a pair of elementary particles - a quark named Quincy who has been separated from his entangled complementary particle, a lepton named Leah. Quincy loves Leah, but they have found themselves torn apart and separated by the very fabric of space and time. The game follows Quincy as he journeys through spacetime to find his lost love. He must use his ability to quantum-tunnel between points in spacetime to navigate these four dimensions, and must rely on his quantum-entangled bond with Leah to help guide him to her; as a result, the gameplay resembles a real-time 2D puzzle game.[1] However, spacetime is not always stable for such small particles, and Quincy's intended path changes in both space and time throughout the game; the player must deal with these fluctuations to move on through each level.

As of now, there are only three simple levels; new levels can be generated arbitrarily easily (given a sensible map of portals, barriers, etc.), but hopefully one can get a sense of the game I was trying to create with just these three primary levels. This app will be available for Android 4.4+ devices on the Google Play store shortly, though I cannot guarantee stable performance and behavior on the breadth of Android devices; see `https://play.google.com/store/apps/developer?id=Abraham+Botros` shortly, or email me at abrahambotros@gmail.com, for the latest version of the app.

2 Motivation

The main reason I selected this topic was that I thought making a simple game would be a fun and engaging way to gain immersive experience in graphics programming for modern/mobile devices. This also seemed like an intriguing area for application of the techniques we have been learning throughout this course. Lastly, I've always had a big interest in gaming and the idea of creating one's own game, so I figured I would try it out a bit here.

In the beginning of the course, one of my main goals was to at least get some general sense of how games are developed, what the main challenges are, and how one would go about creating one's own first game. Though this project was certainly extremely simplified compared to real commercial games, this project was certainly very helpful in greatly improving my familiarity with all these points. This also furthered my understanding of interactive graphics as a whole, which I would consider my particular area of interest in the overall graphics field (interactive software is a general interest for me in computer science as a whole, for that matter). Especially with the rise of incredibly detailed graphical games, hours on hours of gameplay within a single story, and countless computations needing to be made every fraction of a second, modern computer graphics for gaming is an extremely complex, impressive, and intriguing field, and I certainly have even more respect for how difficult game development can be, especially if one were to be dealing with much, much more complex and graphical games than the one created here!

3 Approach

3.1 Planning

As discussed in my project proposals, I initially wanted to construct some basic 3D/first-person game, or some simple side-scrolling/2.5D game. As the TA's pointed out and as soon became clear to me, this was a pretty unrealistic goal for an introductory-level project using no major external game libraries and that needed to be completed in just over a week. I therefore pursued the simplified scenario here, where the game is entirely 2D, the camera is fixed and orthographic, there are limited player actions and world interactions, etc. Overall, I'm fairly happy with the final product for the timeframe I had, and am definitely very happy to have had the

[1] For the related physics, see `https://en.wikipedia.org/wiki/Quark`, `https://en.wikipedia.org/wiki/Lepton`, `https://en.wikipedia.org/wiki/Quantum_entanglement`, and `https://en.wikipedia.org/wiki/Quantum_tunnelling`. Note that I am completely stretching and abusing the actual underlying physics and physics terms for the purposes of a better story, and that in the end there is only a primitive connection between these terms in the games and these terms in real life.

time to at least delve into this subject a bit more!

3.2 Tools

The use of outside tools and libraries for this project was extremely minimal. I used only typical Android OS for app development, and took advantage of the Android system's built-in OpenGL ES (2.0) implementation for graphical display. While I certainly relied on online forums and links for help throughout the way, no other outside libraries or frameworks were used for this project.

All code for this individual project was written in Java/Android, and I did not take advantage of the native NDK/C++ implementation at this time.

3.3 Graphical setup

As previously mentioned, since this was a 2D puzzle-based game, adopting a fixed orthographic camera was a logical and fitting simplification, seem at least in some sense from games like the original "Snake" and "PacMan" games to simple modern mobile games such as Dots [3] and Flow Free [4].

In particular, this involved setting up a viewport with dimensions equal to the integer dimensions of pixels on the device screen during setup. An orthographic projection matrix was computed using these values once, along with a viewing/camera matrix (similar to our coursework). A final model-view-projection matrix was then computed (again, only once for given screen dimensions), stored, and used as the basis for all future transformations and drawing of graphical objects. In particular, [6] was extremely useful in getting this to work properly (essentially, this was helpful in learning how to port our projective transformation code from class to the simpler orthographic transformation here, and on Android instead of in C++). Likewise, [2] and [5] were essential to getting OpenGL ES up and running on Android OS at the start.

As we will see in 3.6, this setup allows us to model almost all basic drawing and transformations as simple translations on a 2D plane, ignoring depth altogether.

At this point, I defined basic shapes (such as triangles and squares, as shown in [2]) and worked on drawing them as in the example figure in the tutorial; see Figure 2.

Figure 2: The initial rendering I worked to emulate to get my implementation off the ground. In [2], the guide works with projection and camera view matrices; since I used an orthographic projection matrix, the initial struggle was simply getting simple triangles and squares to render properly as in this figure.

3.4 User interaction

The next step was to implement some type of user interaction. Since the goal was to move a 2D character in the four cardinal directions, I drew four navigation buttons on top of the rendering surface; all navigation buttons were implemented entirely on the Android side of the code, and commands were then sent into the OpenGL components. Navigation buttons were overlayed on the bottom-left of the landscape screen, for control using the player's left hand. An "action" button (for using portals/activating quantum tunneling events) was placed at the bottom-right of the screen, implemented nearly identically to the cardinal-direction navigation buttons. See Figure 3.

3.5 Textures

Textures were then added onto the player square. At first, this involved a very simple single texture, with no animation and with only a single texture/sprite for the player. This essentially helped localize the player in place, and helped with basic visualization of transformations as discussed in Section 3.6.

Combining the graphical setup (Section 3.3), basic user interaction (Section 3.4), and preliminary texturing discussed here, we see the initial milestone in Figure 3.

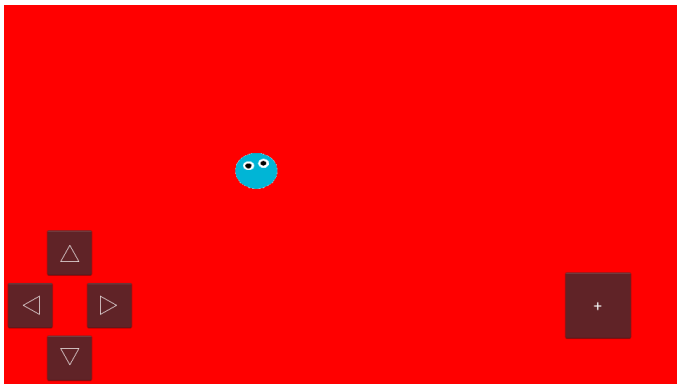


Figure 3: The initial rendering of the game world was incredibly simplistic and basic. It involved simply a background clearing color (red in this image), a single-textured square (with no animation and only one texture/sprite) to represent the player, and the basic navigation and action buttons overlayed on top.

From here, progressive texturing improvements were made to make the game much more visually appealing, including:

- Adding a basic background texture to a single square drawn to cover the whole screen.
- Adding a more aesthetic texture to the player.

These minor changes can be seen in Figure 4.

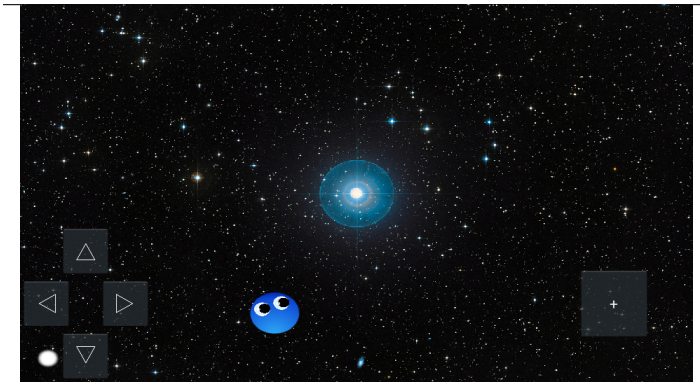


Figure 4: Slight texturing improvements over Figure 3 make a significant difference in aesthetics.

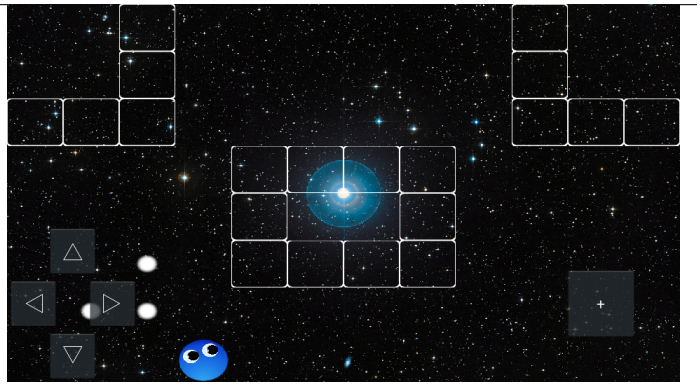


Figure 6: Initial rendering of basic portals and barriers.

Further texturing allowed for a self-orbiting effect (impractical to see in screenshots/images; the player seems to slightly orbit around the center of the cell, with a slightly modified angle at each successive frame to create this effect. Details for this can be found in Section 4.

Lastly, by tracking the direction that the player last moved in, some effect of navigation direction was also incorporated using textures. In particular, the player sprite (Quincy) is drawn looking in the cardinal direction of movement during each motion; see Figure 5.

As a side note, other texture resource links (used for creating final textures for the portals, barriers, and backgrounds) are included in the code submission of this project, for reference.

3.6 Transformations

As discussed in Section 3.3, the restriction of rendering to a 2D plane using orthographic projection greatly simplified the rendering of shapes and, in particular, transformations. With only two degrees of freedom (the x and y directions in the image), transformations essentially boiled down to translations in the x and y directions to simulate movement in the game world. Translation was also used to simply place objects at various places on the 2D map, even if the objects were not moving.

To place an object/square at a particular location for a given frame, the base coordinates of that square are precomputed according to the screen dimensions, and then at the time of drawing, we apply the model-view-projection matrix from Section 3.3 and the translation matrix from the object's desired (x, y) coordinates. Thus, we can move the player as in Figure 5, and can also place non-moving objects as in Figure 6.

3.7 Game world interaction

To actually create some gameplay and not just have navigation, some game objects needed to be added. In particular, there were two types of objects that needed to be implemented:

- **Portals**, which reside at a specific stationary game location and teleport the player instantly to a designated location if the player triggers the portal.
- **Barriers**, which also reside at a specific stationary game location, but that simply prevent the player from walking to that game location.

Both types of objects were also extensions of the basic rendering-square object, with the main difference from the player object being that neither can move like the player can. As discussed in Section 4, slightly more involved texturing allowed the portals to be drawn based on their characteristics and animated on use, while barriers were textured based on the current level.

The initial barrier and portal object renderings can be seen in Figure 7.

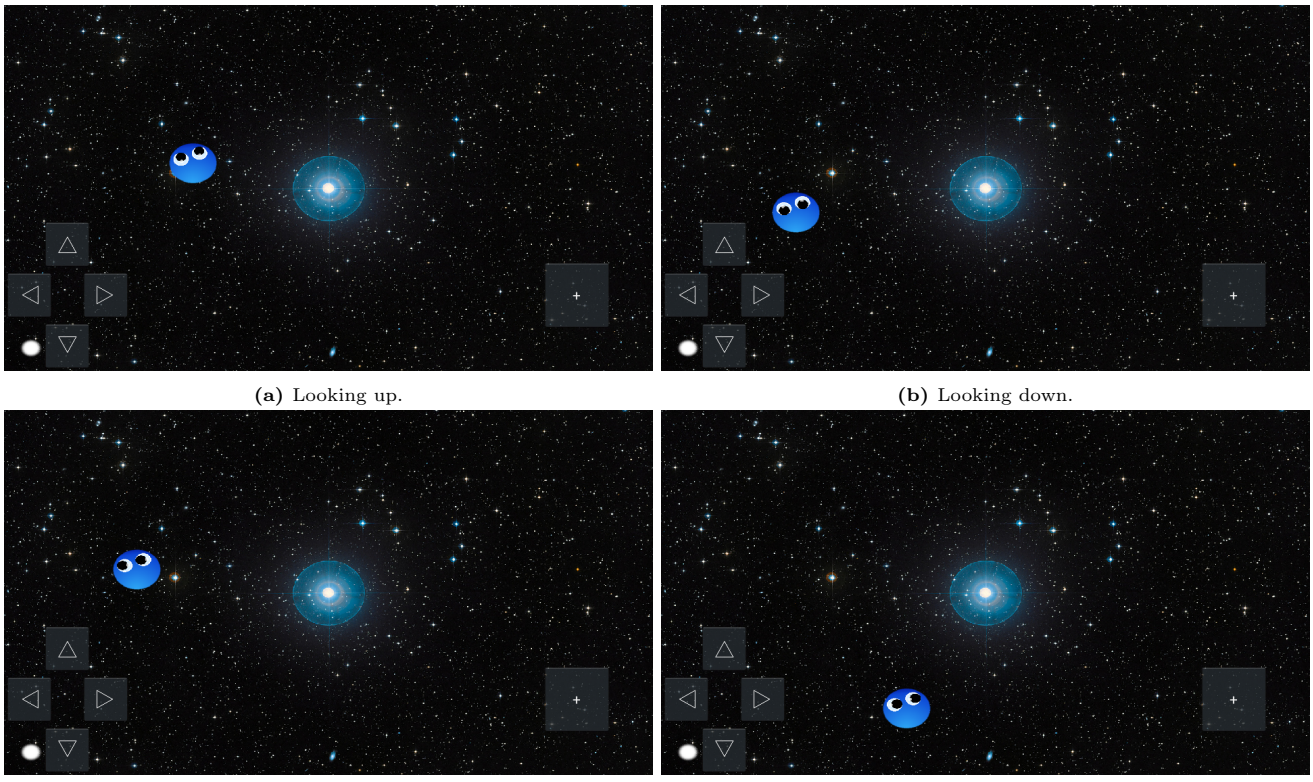
Some notion of levels was also created here, with different levels having different hard-coded layouts of game objects and different background and barrier textures.

3.8 Game logic

A decent amount of game logic was also implemented even for this basic game. Some more important aspects included:

- Checking and rounding of player coordinates based on movements. See Section 4.
- Preventing the player from traveling onto/through barrier locations.
- Handling portal teleportation from one location to another.
- Handling "goal" portal selection, and teleportation through to subsequent levels. See below.

In particular, one major game element was the switching of "goal" portals every few seconds (the duration differs per level). Every few seconds, a new "goal" portal is chosen pseudo-randomly and redrawn with the distinct pink texturing to represent a portal that would lead Quincy closer to Leah. There is only one goal portal active in a given level at any given time. If this goal portal is activated by the player in time, the player completes the level and moves on to the



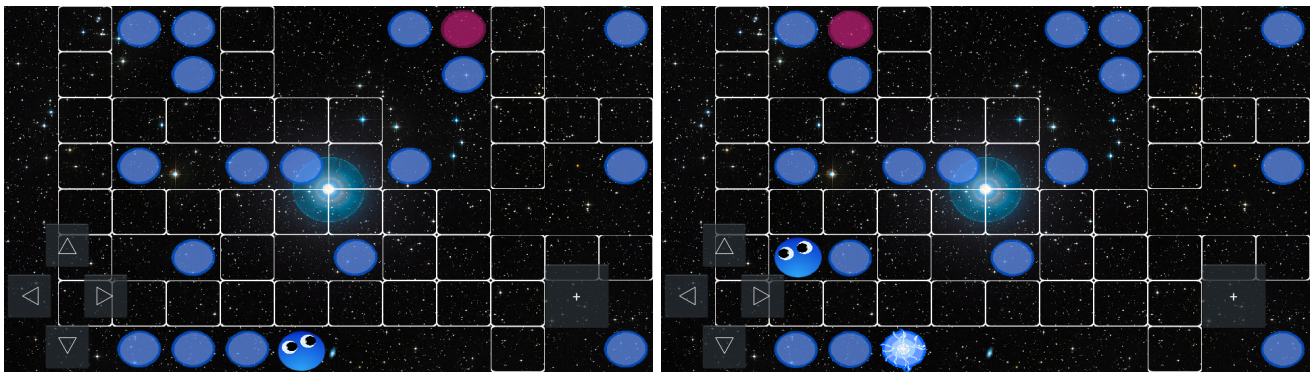
(a) Looking up.

(b) Looking down.

(c) Looking left.

(d) Looking right.

Figure 5: Example images showing the player sprite looking in different directions based on the player's most recent movement.



(a)

(b)

Figure 7: Example images showing the basic portals and barriers being rendered. (b) shows a subsequent frame following (a) after the player has teleported through the bottom-center portal he/she was standing on in (a); note that said portal is currently being animated in (b) because it was just used.

next.

I also needed to strictly track the phase of the app (whether in the initial story-telling - see Section 3.9 - or the actual game, or outro. This was particularly complicated by the hassle of communication and timing between the UI/application thread and the underlying OpenGL/rendering thread (see Section 4).

Another main point of logic was handling frame rate. I used the simplistic approach described in [7], where the OpenGL thread sleeps the appropriate amount to ensure that frames are rendered as close to the desired framerate (30 FPS, here) as possible. This is likely not an optimal approach since sleeping a thread is usually undesirable, but this proved fairly simple and effective.

3.9 Story-telling

Lastly, after the essentially game had been put together, an element of story-telling and visuals was added. This primarily involved adding introductory/story slides (on the Android OS side of things) before starting the game, which the user can click through before starting the first level. Likewise, an outro slide was added if the user completes all three levels.

Along with the added texture animations previously described, adding more engaging backgrounds, coloring of portals, and barrier textures certainly helped in bringing the game to life a bit more. The final levels can be seen in Figure 8.

4 Challenges

4.1 Challenges conquered

4.1.1 Textures

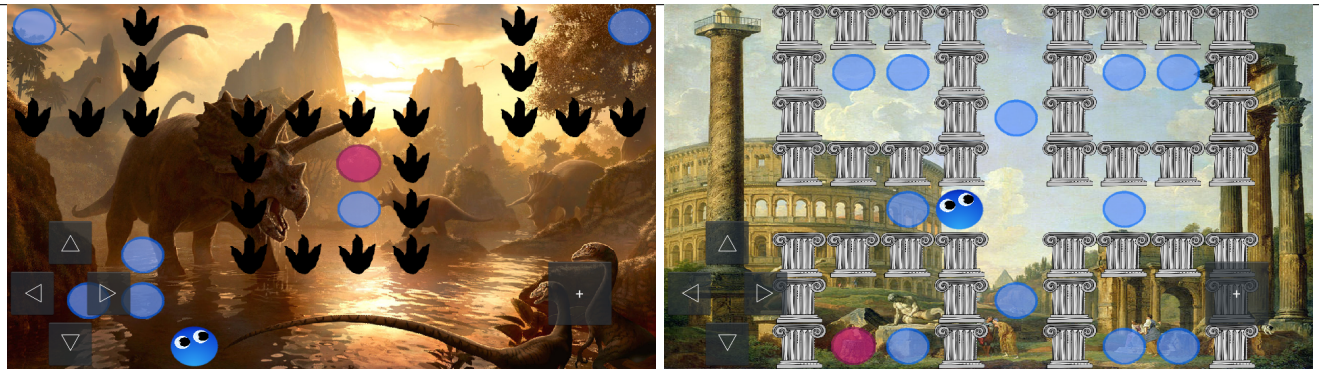
Numerous challenges arose as I made my way through the development of this game. The primary one was by far animation using textures. This involved creating various sprites for each desired animation (such as the player orbiting within a cell, the player facing different directions based on navigation direction, differing portal colors based on the "goal" property, and a portal being animated after use). The whole idea of sprite animation was entirely new to me. Resources such as [1] were particularly helpful in understanding how to draw textures using OpenGL ES on Android.

A decent amount of time and effort was put into writing generalizable code for handling any number of sprites/rows/columns, which certainly proved useful for dropping in new textures for different objects as I proceeded.

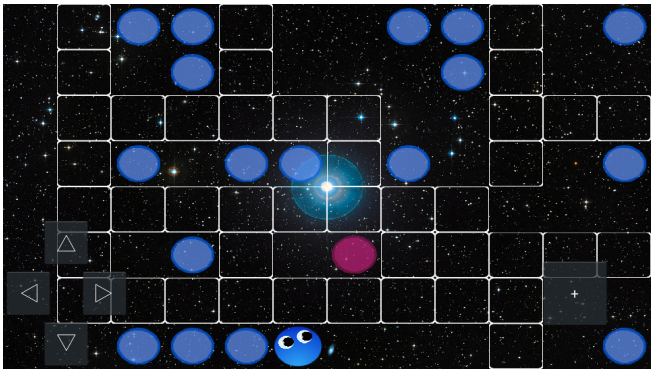
In particular, the player's self-orbiting animation was handled by creating 8 sprites for each direction to correspond to 8 consecutive frames (and repeating frame sets). The player circle was offset by the appropriate progressive amounts to give the orbiting effect over subsequent frames. This offset was done offline, so that the orbiting textures were fully ready to be displayed at runtime with no modification or special handling of texture coordinates, for example. Player direction-dependent rendering was handled in much the same way. Both of these can be seen in Figure 9.

Figure 9: Player sprites used to handle both self-orbiting animation and direction-dependent rendering. It might be hard to see, but for each couple of rows (corresponding to a specific looking direction), the player is offset by varying amounts of each of the 8 sprites, giving the self-orbiting animation.

A similar approach was taken for animating portals, both based on the "goal" property and based on use. The sprites used for the portals are shown in Figure 10 and were primar-



(a) (b)



(c)

Figure 8: Screenshots of final levels with all texturing and animations implemented. The various levels represent Quincy's journey through space and time to find Leah. The first level corresponds to prehistoric times, where the dinosaurs still roamed the Earth. The second corresponds to times of the grand historic civilizations such as Rome. The third corresponds to a futuristic/space setting, where Quincy is making his final search for Leah in the vastness of space.

ily created using incremental rotations offline of a base portal sprite. This was a bit more complicated, as portals are animated for about a second after the player uses them (and while the player has already begun moving at the teleported-to location). To tackle this, a separate timer/frame-counter was started each time a portal was used, and the portal was given a modified texture based on this timer/counter and the portal's "goal" property. One last complication was getting the "goal" portal to animate after the player steps through it to end a level; exceptions were made to make sure the level persisted (while only showing the "goal" portal) after the player had already completed it, for some specified amount of frames/loops.

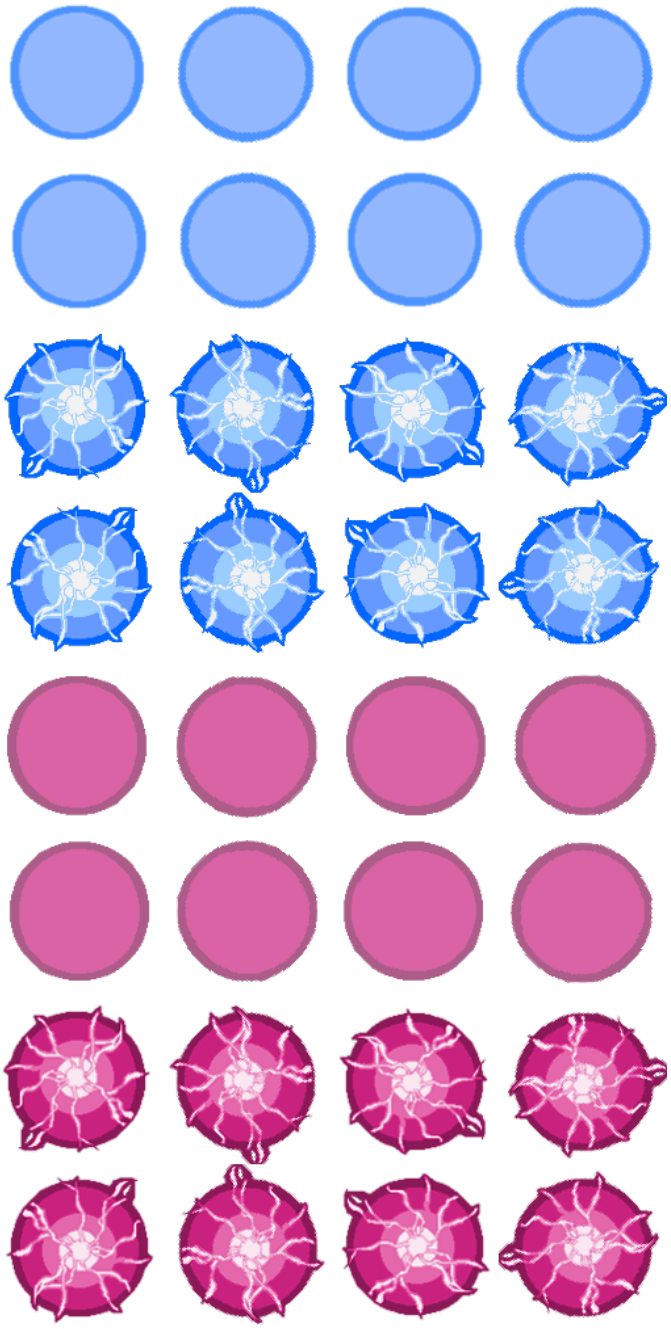


Figure 10: Portal sprites used for handling the "goal" property (pink is a "goal" portal, blue otherwise) and for animating when used by the player.

4.1.2 Coordinates

Another minor complication was handling of screen coordinates. At first, I was handling game locations as the intuitive grid of integers, where each tile (approximately the width and height of the player sprite) had unit length 1. However, this proved problematic for animating incremental movement, as this involved dealing with floats (stepping from 1.0, to 1.2, to 1.4, and so on until 2.0) for intermediate steps and brought with it a whole slew of the expected numerical error problems and inconveniences. My solution was to instead deal with a grid of integers multiplied by a factor of 10 (such that the location (1.2, 2.0) would instead be computed as (12, 20)), which eliminated any need for decimal-valued numbers and was much, much cleaner and effective overall. The only time these "tile" coordinates were converted to pixels was in the very last step when computing the transformation matrix for each square object.

4.1.3 User interaction, navigation

A related issue was making sure user interaction/navigation worked properly in-game. For example, this involved making sure the user could only trigger more movements from proper tile corners, and not while in intermediate locations while moving between rounded tile coordinates (i.e., no additional movement should be queued up if the user is at (15, 20) while moving between (10, 20) and (20, 20)).

4.1.4 Threads

One other problem was dealing with the differing threads used by the Android system when running this program. In particular, since all OpenGL commands must be run on the separate OpenGL thread, while user interaction (and things like the introductory slides, handling player navigation, and the outro slides) were all handled in the Android UI thread. Thus, something as simple as creating the game level and game objects when the user finishes the introductory slides was slightly complicated by having to flip switches and send messages between the threads. Having a dedicated game-engine-like object (the GameEngine.java class in my implementation) helped coordinate things between these threads, but this certainly could still be cleaned up and streamlined more.

4.2 Challenges remaining

Still, a variety of issues still remain that would be addressed in any future work on this game. This includes:

- Primarily, adding some sort of timer/score system to the game to make it more engaging. This would, for example, penalize players for sitting and waiting for a nearby portal to become a "goal" portal, and would instead encourage them to actively navigate towards the current goal at any given point in time. This could also provide a threshold for success and failure in each level; for example, the player might have to reach the goal by a specific number of seconds after the start of the level, otherwise the player fails that level and must restart.
- Adding more levels would certainly be necessary. It would be interesting to have various underlying possible portal networks that are chosen from randomly each time a given level is instantiated, so that the player has a new challenge each time while remaining at approximately the same difficulty since playing the same level.
- Minorly, some transformation matrices (for non-moving objects) could be stored instead of computing them each time. This would save a few clock cycles between each rendering.
- Better handling of onPause/onResume system callbacks, along with screen orientation/dimension changes. None of these are adequately and thoroughly handled at the moment, leading to unpredictable behavior when the device sleeps and resumes, or when the orientation of the device changes (currently, the app restricts the device to stay in strictly landscape mode, so this is some-

what hack-ily handled).

5 Lessons learned

I learned numerous things about gaming - what learned about gaming - screen timing, cross-thread communication, variable handling, etc. all difficult

6 Conclusion

In conclusion, this was a fun and engaging project, and I felt it was a great first experience with mobile gaming. I am happy I got the chance to work on this, and hopefully can continue work on this project in my spare time in the future. I felt that overall this was a success, even though the final game ended up quite simplistic, short, and simple. It was interesting to see the development of a small graphical story at my own hands, and I am happy I got to share in Quincy's and Leah's little adventure.

I learned numerous things about gaming and computer graphics as a whole in the process. In particular, I got a better sense (if even just an inkling) of the difficulty and breadth of problems that game developers must deal with when creating games, such as screen timing, cross-thread communication, and handling of numerous game-specific variables and states. I also got a better sense for how implementations of games on mobile devices work, and some of the special care that must be taken in mobile settings.

In general, working with such games and interactive graphics was very intriguing, and I certainly now have more desire to continue to explore this space. I feel that this project was a very fun and immersive way to get my hands wet, and I hopefully will have time to pursue more in the future. Since this app will be soon posted on the Google Play store, I hope you enjoy this little story of Quincy and Leah too! Thank you!

7 Special thanks

Special thanks to my girlfriend, Michelle Karpovich, for helping me create and choose the textures used in this project. Thank you!

Also, more special thanks to the authors of the following links and graphics:

- Intro slide background: `https://upload.wikimedia.org/wikipedia/commons/0/0d/Hubble_ultra_deep_field_high_rez_edit1.jpg`
- Intro slide arrow: `http://www.4kingdoms.com/arrow.jpg`
- Dinosaur background: `http://thedinosaursofearth.blogspot.com/`
- Rome/Colosseum background: `http://fineartamerica.com/featured/the-colosseum-and-other-monuments-giovanni-paolo-panini.html`
- Rome/Colosseum pillar for barrier texture: `https://openclipart.org/detail/29053/capitello`
- Space background: `http://www.space.com/27728-around-beta-pictoris-space-wallpaper.html`
- Portals: `http://www.deviantart.com/art/Dragon-Ball-Z-LSW-Attack-Sprites-326401952`
- Outro slide heart: `http://claudias-family.blogspot.com/2011/09/heart-wanted.html`

References

[1] Android lesson four: Introducing basic texturing. `http://www.learnopengles.com/android-lesson-four-introducing-basic-texturing/`.

[2] Displaying graphics with opengl es (android developers guide). `http://developer.android.com/training/graphics/opengl/index.html`.

[3] Dots: A game about connection (google play store). `https://play.google.com/store/apps/details?id=com.nerdyoctopus.gamedots`.

[4] Flow free (google play store). `https://play.google.com/store/apps/details?id=com.bigduckgames.flow`.

[5] Opengl es (android developers guide). `http://developer.android.com/guide/topics/graphics/opengl.html`.

[6] A real open gl es 2.0 2d tutorial part 1. `http://androidblog.reindustries.com/a-real-open-gl-es-2-0-2d-tutorial-part-1/`.

[7] Stackoverflow - how to limit framerate when using android's glsurfaceview.rendermode_continuously? `http://stackoverflow.com/questions/4772693/how-to-limit-framerate-when-using-androids-glsurfacevi lq=1`.