Proceedings of the 17th Linux Audio Conference (LAC-19), CCRMA, Stanford University, USA, March 23–26, 2019

CREATING A SONIFIED SPACECRAFT GAME USING HAPPYBRACKETS AND STELLARIUM

Angelo Fraietta

UNSW Art and Design
University of New South Wales, Australia
a.fraietta@unsw.edu.au

Ollie Bown

UNSW Art and Design
University of New South Wales, Australia
o.bown@unsw.edu.au

ABSTRACT

This paper presents the development of a virtual spacecraft simulator game, where the goal for the player is to navigate their way to various planetary or stellar objects in the sky with a sonified poi. The project utilises various open source hardware and software platforms including Stellarium, Raspberry Pi, HappyBrackets and the Azul Zulu Java Virtual Machine. The resulting research could be used as a springboard for developing an interactive science game to facilitate the understanding of the cosmos for children. We will describe the challenges related to hardware, software and network integration and the strategies we employed to overcome them.

1. INTRODUCTION

HappyBrackets is an open source Java based programming environment for creative coding of multimedia systems using Internet of Things (IoT) technologies [1]. Although HappyBrackets has focused primarily on audio digital signal processing—including synthesis, sampling, granular sample playback, and a suite of basic effects–we created a virtual spacecraft game that added the functionality of controlling a planetarium display through the use of WiFi enabled Raspberry Pis. The player manoeuvres the spacecraft by manipulating a sonic poi[1], which is usually played in the manner shown in Figure 1. The poi contains an inertial measurement unit (IMU), consisting of an accelerometer and gyroscope; and a single button. The goal of the

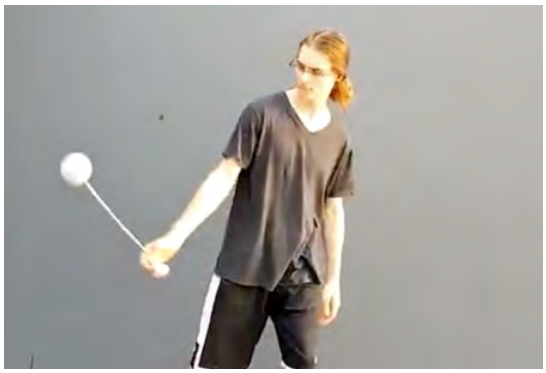

Figure 1: *The conventional way of playing a sonic poi.*

game is for a player to choose an astronomical object, for example a planet or star, and to fly towards that object. This enables the player to view other objects, including planets, moons, stars and galaxies in the field of view. For example, Figure 2 shows how the player might view Saturn from Earth, while Figure 3 shows how the player may view Saturn from their spacecraft. The sonic poi generates sound that is indicative of the player's field of view. Additionally, the poi provides audible feedback when the player zooms in or out.

Figure 2: *Saturn viewed from the ground from Stellarium.*

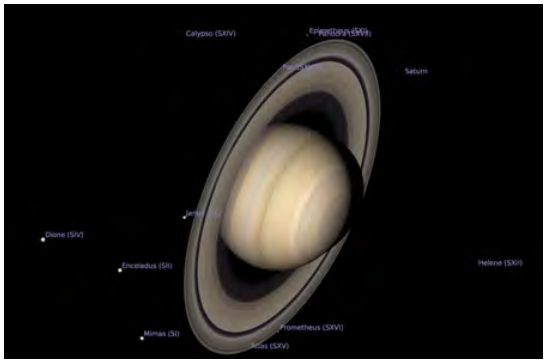

Figure 3: *A closer view of Saturn from Stellarium.*

The University of New South Wales required a display for their open day to showcase some of the work conducted in the Interactive Media Lab. The opportunity to develop an environment whereby visitors could engage with the technology we were developing would not only facilitate attracting possible future students, it was also a way to develop and test the integration of various research components we were conducting. Many managers and business seek to engage new customers through gamification [3]—in this case, prospective customers were potential students. Furthermore, research indicates that visualisation and interpretation of software behaviour de-

[1] "Poi spinning is a performance art, related to juggling, where weights on the ends of short chains are swung to make interesting patterns." [2, p. 173]

veloped as part of a game is more memorable, which facilitates locating errors or developing methods for improvement [4]. Developing a game, therefore, would not only engage the visitors, it would provide us with a more memorable way of seeing how our system was behaving.

The technology to develop the game required two different versions of Raspberry Pi, installation of planetarium software onto one of the Pis, and the creation of a Java API to join the different systems. This paper details the strategies and techniques to integrate the different technologies and describes some of the workarounds for unresolved issues. We also discuss the goals, rules and rewards used to define the game and the methods we used to entice prospective players. Finally, we lists areas where the research can be extended.

2. BACKGROUND TO RESEARCH

The research was inspired by a previous project developed by one of the authors that correlated what a viewer saw in the night sky through binoculars with data obtained from on-line astronomical data catalogues [5]. One installation, which was conducted in conjunction with the Newcastle Astronomical Society on one of their field viewing nights, was particularly successful [6]. More than twenty members of the public were enticed into viewing the night sky through high powered binoculars while sound that was based on data from the stars they were viewing was playing through loudspeakers on the field.

Another set of performances was conducted with an improvising ensemble that featured various astronomical photos displayed as a slide show [7]. The stellar data was mapped as MIDI and successfully functioned as inspirational impetus for the performers, but was unsuccessful from an astronomical point of view. First, the ability for viewers to look through the equipment was directly dependant upon the weather. One performance, for example, had a night sky complete with thick black cloud, heavy rain and lightning. Moreover, when the weather was favourable for viewing, the audience were often content to just watch the performers rather than venture out of their chairs to view through the binoculars [5]. The audience feedback from the was that although they really liked the slide show, many were unaware that the binoculars were even there for viewing. Instead of providing a slide show at the next performance, an improvisation using Stellarium from a laptop computer was used on the screen. The audience's response was extremely favourable, inspiring the idea of using Stellarium as a visual stimulus instead of binoculars.

2.1. Raspberry Pi

The Raspberry Pi was originally developed in 2011 [8] for education by the Raspberry Pi Foundation, a UK based educational charity [9][10]. The Raspberry PI has a very large user base and a significant number of plug in sensors available for it [11], and supports a 128GB SD card, which can be used to store more than 200 hours of high-quality audio. The Raspberry Pi foundation officially supports a derivative of the Linux distribution Debian known as Raspbian [12]. Raspbian's inclusion of compilers, support for multiple coding languages, and the ability to run multiple programs provides the flexibility that enables a system to expand as an interactive platform as newer technologies become available. The game project used two different versions of Raspberry Pi and Raspbian. The sonic poi required a small form factor, low power consumption but did not require a GUI, and consequently, Pi Zero running Raspbian Stretch Lite was selected. The device used to display the graphics required significantly more power but did not have size restrictions, so a Raspberry Pi B+ running the desktop version of Stretch was selected for this.

2.2. HappyBrackets

HappyBrackets commenced as "A Java-Based remote live coding system for controlling multiple Raspberry Pi units" [13] where a master controller computer sent pre-compiled Java classes to selected Raspberry Pi devices on a network. Unlike the Arduino sketch, which is effectively a single program [14], the HappyBrackets composition is not a standalone executable program. The HappyBrackets core has a thread that listens for incoming bytecode classes, and after receiving the class, executes the new class's functionality through a Java interface. This allows for multiple concurrent compositions that can be easily created or updated during composition or the creative coding performance [1]. This research was extended with the development of the Distributed Interactive Audio Device (DIAD) [15], which contained an IMU consisting of an accelerometer, gyroscope and compass. The devices were handled by the audience and incorporated into the environment. The DIADS not only responded to user manipulation, they also responded to one another. Furthermore, DIADS were configured to automatically connect to the wireless network, and once a DIAD came into range of the network, became a part of the DIAD multiplicity. The main focus of this development was the implementation of a reusable platform that allowed creators to easily develop interactive audio and easily deploy it to other devices. Although HappyBrackets runs on many embedded platforms, the main research has been with the Raspberry Pi, primarily due to the availability and low cost of the devices. HappyBrackets is licensed under the Apache License 2.0[2] and is available through Git Hub[3].

A prebuilt disk image—which contains the Java Virtual Machine (JVM), the I2C drivers to enable access to the IMU, and libraries to access the GPIO—enables users to flash an SD card and start using HappyBrackets without ever having to connect their device to the Internet. The licence for the Oracle JVM, however, appeared to prohibit embedding the Oracle JVM into a prebuilt image and was therefore legally problematic. We found that the AZUL Zulu JVM was available under the GNU GPLv2 licence[4], enabling an embedded distribution within an image. Medromi et al. conducted a study that compared the two JVMs [16]. Their tests revealed that Zulu created more threads and classes than Oracle, indicating that Zulu probably used more memory, making it more susceptible to garbage collection issues. Furthermore, their tests showed that Zulu also used a greater percentage of CPU, indicating greater power consumption. The report, however, did not detail the difference in performance speed between the two JVMs. Our own initial tests did not show any difference between the two JVMs and there was no noticeable performance degradation, however, this is an area we still need to research. It is possible to change the default JVM used in the Raspberry Pi from the terminal, which would make switching between JVMs when performing comparative tests relatively easy.

[2] www.apache.org/licenses/ [accessed November 2018]

[3] github.com/orsjb/HappyBrackets [accessed November 2018]

[4] www.gnu.org/licenses/old-licenses/gpl-2.0.txt [accessed November 2018]

2.3. Stellarium

The advancement of computing power over the last two decades has made the availability of planetarium software available on both desktop computers and mobile devices commonplace. Moreover, many of these software packages—including *RedShift*[5], *SkySafari*[6], *StarMap*[7], *The SkyX*[8], and *Stellarium*[9]—have become valuable tools for astronomers. They facilitate the identification of objects and in the planning of viewing and astro-photography sessions by enabling sky simulation for any particular location, date and time [17].

Stellarium is an open source software project distributed under the GNU General Public Licence with the source code available through Git Hub[10] . Stellarium functions as a virtual planetarium; calculating positions of the Sun, moon, stars and planets based on the time and location defined by the user. Moreover, the viewing location does not even need to be on Earth. For example, Figure 4 displays Stellarium rendering Jupiter viewed from its moon Io.

Figure 4: *A simulation of Jupiter viewed from Io.*

Stellarium is used by both amateur and professional astronomers, and is used by the European Organisation for Astronomical Research in the Southern Hemisphere to facilitate distribution and sharing of visual data among scientists [18]. Stellarium has a very high quality graphical display, supporting spherical mirror projection that can be used with a dome [19]. Stellarium is used in many schools and museums because it is both scientifically accurate and visually engaging [18]. Moreover, it is suitable for demonstrating basic through to advanced astronomy concepts [18]. Stellarium has a built in library of 600 000 stars, with the ability to add an additional 210 million [19]. Moreover, Stellarium can display constellations from several different cultures and has labels translated to more than 40 languages, making Stellarium both culturally aware and inclusive [18].

Although it is quite straightforward to control Stellarium using a keyboard and mouse, there are many plugins that allow third party integration with the software. The plugin we were particularly interested in to control Stellarium was the Remote Control, which enabled control of Stellarium through HTTP [21]. Stellarium also contains a powerful scripting engine that enables one to program and run complete astronomy shows. The scripts, written in JavaScript, control Stellarium through a series of objects that represent the Stellarium application components [20].

[5] www.redshift-live.com [accessed November 2018]
[6] www.southernstars.com [accessed November 2018]
[7] www.star-map.fr [accessed November 2018]
[8] www.bisque.com [accessed November 2018]
[9] stellarium.org [accessed November 2018]
[10] github.com/Stellarium/stellarium [accessed November 2018]

3. RELATED WORK

Video games rose from obscurity in the 1970s, into a video arcade industry grossing $8 billion dollars in 1982 [22, p. 88]. The video game moved from the arcade into the home with Nintendo and Atari game consoles [22, 23]. Iconography games like *Space Invaders*, *Defender*, *Spaceward HO!* and *Star Wars* were often replaced with interactive games that became more realistic [23]. Wolf suggests that there are more than forty different genres of video games [23], however, we were only particularly interested in the "Training Simulation" genre.

One study showed that video game expertise developed over long-term playing had a beneficial effect on the spatial skills in the player, supporting the hypothesis that "video expertise could function as informal education for the development of skill in manipulating two-dimensional representations of three dimensional space" [22, p. 93]. The aerospace industry has employed training simulators for many years, with the advancement in virtual reality environments leading to the availability of a new technology known as "serious gaming" [24, p. 655]. This technology exploits popular high-quality computer games, making it available via Software Development Kits (SDKs) to developers of "serious"[sic] applications such as defence, surgery, education and aerospace [24, p. 686].

One particularly interesting training simulation project was a prototype environment for training astronauts in a simulated zero gravity environment for the purpose of controlling and handling objects [25]. Rönkkö et al. noted that astronauts discovered using a laptop in a zero gravity environment was completely different to using it on Earth, and that the whole concept of a laptop computer in a zero gravity environment was questionable [25, p. 183].

There have been various implementations of third party integration with Stellarium. Although it is possible to remotely control a telescope using Stellarium as the master controller [26], some researchers have developed projects whereby Stellarium becomes the slave. Tuveri et al. developed two planetarium control systems for driving Stellarium on a Laptop computer [27]. They extended the Stellarium code in order to send it application messages before the Remote Control plugin was available in the standard Stellarium distribution. One interaction implementation was through a touch screen, while the other was through a Kinect gesture controller [27].

The Remote Control Stellarium plugin was developed by "Florian Schaukowitsch in the 2015 campaign of the ESA Summer of Code in Space programme" [20, p. 110], and was used for a visual art installation in the MAMUZ museum for pre-history [21]. The installation, *STONEHENGE. A Hidden Landscape*, consisted of a single computer driving five projections onto a 15x4m curved screen.The presentation was automated with a Raspberry Pi that triggered a script via an HTTP request every twenty-five minutes via a cron job. This Remote Control plugin is now a standard part of the Stellarium installation. This use of both scripting and HTTP control was the mechanism we employed in our game.

4. DEFINING THE GAMIFIED EXPERIENCE

One of the intentions of creating the gamified environment was to engage visitors. In the gamified experience, four parties are involved: players, designers, spectators, and observers [28]. The key to a de-

veloping successful gamified experience is to identify who the parties are and how to engage them for the purpose of creating a positive and memorable experience [3], each with different levels of involvement or immersion [28]. Players were the visitors who physically controlled the virtual spacecraft, and in a sense, were the competitors and highly immersed in the experience. Spectators were people who do not directly compete in the game, but instead, influenced the game indirectly by encouraging the player and were also highly immersed in the experience. Observers were other visitors in the space that were passively involved and had no direct impact on the game. They were, however, mildly involved and often moved to become players or spectators [28].

Research indicates that the three main factors in developing an enjoyable game were challenge, fantasy and curiosity [29]. We provided challenge in that we set a goal that had increasing levels of difficulty. As the user was closer to the planet, the spacecraft became more difficult to control.

We utilised fantasy in that we implement two modes of play: terrestrial and spaceship. Terrestrial mode allowed the player to use gravity in a familiar way, provided wide fields of view that showed large amounts of sky and provided course control. Spaceship mode showed less fields of view, displaying significantly less sky and provided finer control; however, the player was not allowed to use gravity in their control. We enabled the player to zoom in and out by performing a quick twist action of the ball around the string. If the gyroscope pitch value exceeded the set threshold, the field of view would change, simulating a zoom in or out. When the user changed their field of view to less than 30 degrees, the play mode went from terrestrial to spaceship. We provided an audible feedback that sounded like a zipper when the level of zoom was changed.

The only controls available at the time on the poi were accelerometer and gyroscope [11], while the only feedback was audio generated by the poi and the Stellarium display. In the same way that a laptop could not be used conventionally in a zero gravity environment [25], a player would be unlikely to control the game successfully using the poi by spinning it around their body [2]. Figure 5 shows the poi with three axes of accelerometer and gyroscope on the left and right respectively.

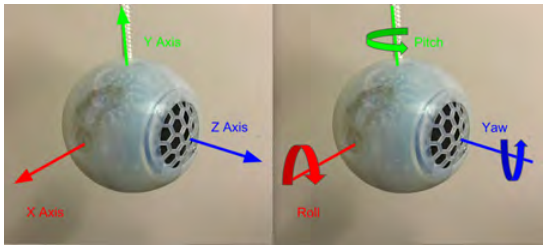

Figure 5: *Sonic Poi accelerometer and gyroscope input.*

In terrestrial mode, we wanted to simulate a viewer on the ground lifting and turning their head to view the sky as one would on Earth, which is essentially increasing the altitude and rotating the azimuth. The player "lifts their head" by raising the ball of the poi in an arc, using the point where the player holds the rope as the centre, and measuring Y axis acceleration through the IMU in the poi. Rotating the viewer's head was simulated by detecting the pitch value of the gyroscope, as shown on the right side of Figure 5. Gyroscope values only change while the object is rotating, whereas gravitational accelerometer values are maintained when the object is stationary.

In the spaceship mode, we wanted to simulate the player navigating through space in a zero-gravity environment. The yaw and the pitch were used as input, whereby the user had to roll the ball in their hands to move the display. This was completely foreign to users at first because there was no haptic feedback, nor any sense of grounding for the user or the control. In a sense, it was similar to balancing on a ball in space because you could not fall off—you would just float in an unintentional direction. Furthermore, it was not easy to detect which axis was which because the poi was a ball shape. Furthermore, rotating one axis would affect the cognition of the other axis. Consider a player in Figure 5 rotating the ball forward around the X axis with the poi producing a positive yaw. If the player then turned the poi 180 degrees around the string, rotating the ball forward again would now produce a negative yaw, which would mean the screen would start moving in the opposite direction to what they experience a moment earlier. The result was that controlling the display required constant mental adjustment, which we suggested might simulate to some degree the sense of strangeness an astronaut may feel controlling objects in outer space [25, p. 183].

In order to run an attractive and engaging display that would trigger the visitors' curiosity when they entered the room, we ran Stellarium scripts that functioned as standalone astronomy shows. We invited visitors to manipulate the poi and watch the display move while the script was running. When we saw they were interested and enjoyed the novelty of interacting with the display through the poi, we offered them the opportunity to start from Earth and navigate to one of the planets in our solar system. As they zoomed in closer to Saturn, they became quite excited when they saw the rings and realised that they could also see Saturn's moons. For those who were particularly enthusiastic, we suggested finding Jupiter next, informing them that they would also be able to see the four Galilean moons that night at home with a standard pair of binoculars. We also asked them to imagine that rolling the ball to control their movement might be as strange as moving about in a zero gravity environment. Although a few of the players gave up after a few minutes, the majority of players continued for more than ten minutes, had a lot of fun, and exhibited a sense of achievement in being able to navigate into outer space.

5. DEVELOPMENT

The system was originally developed as a tool for evaluating the performance, behaviour and suitability of networked control of Stellarium as part of a potential interactive audio visual artwork. We intended to calculate the azimuth and altitude position in space calculated from the rotation and manipulation of poi. These values would be used as input to Stellarium on another device, sent via the network, which would then display the sky based on those values. Additionally, we sent commands to change the field of view on Stellarium, which effectively acted as a zoom function. The poi also played audio as a series of ten uniformly distributed pseudorandom sine waves between 500 and 550Hz, giving a sense of cosmic background microwave noise.

```
float freq = hb.rng.nextFloat() * 50 + 500;
Envelope envelope = new Envelope(1);
WaveModule soundGenerator = new WaveModule();
```

[11] The button control was added to the poi later.

```
soundGenerator.setFequency(freq);
soundGenerator.setGain(envelope);
soundGenerator.connectTo(masterGain);
return  envelope;
```

A metronome iterates though each of the envelopes, adding segments that cause each frequency to momentarily pop out of the background as a beep.

```
hb.createClock(5000).addClockTickListener((
    offset, this_clock) -> {
Envelope e = envelopes.get(envelopIndex++ %
    TOTAL_OSCILLATORS);
final float LOUD_VOL = 20;
final float LOUD_SLOPE = 20;
final float LOUD_DURATION = 200;

e.addSegment(LOUD_VOL, LOUD_SLOPE);
e.addSegment(LOUD_VOL, LOUD_DURATION);
e.addSegment(1, LOUD_SLOPE);
});
```

As the user zooms in, the metronome becomes faster, increasing the beep rate, generating a sense of sonic tension.

5.0.1. Starting Stellarium

The first challenge was starting Stellarium on the Pi from within HappyBrackets. HappyBrackets has a simple facility to execute shell commands or create processes through both the Java Runtime exec and the ProcessBuilder [30]. We attempted a script to run Stellarium from a process command, which ran successfully when executed from a terminal; however, we could not get HappyBrackets to run the script after each fresh reboot of the device—the program was unable to access the display. Interestingly, If we killed the JVM and the started HappyBrackets again from a terminal, then Stellarium started from within HappyBrackets with no problem. The problem was that the HappyBrackets installation script had configured the Raspberry Pi to automatically start the Java application when the device first boots by executing a script in /etc/local.rc as defined in the Raspberry Pi documentation[12]. In order to run GUI programs from Java, the Java program needs to be started when the desktop starts, which was effected by moving the script command to /.config/lxsession/LXDE-pi/autostart[13]. The HappyBrackets installation scripts were consequently modified to detect whether a desktop version was used, and added the HappyBrackets start-up script command accordingly.

5.0.2. Controlling Stellarium

Examples of controlling Stellarium through the Remote Control API were provided on the plugin developer page[14], which made use of the *cURL [sic]*[15] command line utility[16] and executed via an SSH terminal connection to the Pi. Although we did not intend to use curl in our actual program because Java has its own networking interface, curl was extremely useful for examining and diagnosing through the terminal. Querying the state of Stellarium was performed by issuing a curl *GET* command. For example, executing the following command in the SSH terminal retrieves the current view direction of Stellarium as a JSON encoded string.

```
curl -G http://localhost:8090/api/main/view
{"altAz":"[0.954175, 9.54175e-06,
    0.299249]","j2000":"[0.240925, 0.147495,
    -0.959271]","jNow":"[0.241334, 0.148053,
    -0.959082]"}
```

Setting the position of Stellarium is executed with the curl *POST* command, with the parameters added as JSON parameters. Executing the following command would set the display to horizontal by setting the altitude to zero.

```
    curl -d 'alt=0' http://localhost:8090/api
        /main/view
```

Having tested the functionality using curl through the terminal, we implemented calls using the standard Java URL connections [31]. We sent control message from the poi via UDP to the slave using HappyBrackets and then immediately sent the HTTP message on the slave to Stellarium. We found that although the message arrived from the poi to the slave in less than a few milliseconds, the time to execute the post message on localhost, be actioned by Stellarium, and then return typically took between 80 and 120 milliseconds. This produced accumulative latency when the player continually moved the poi. The accelerometer and gyroscope typically update every 10ms, so constantly rotating the device for two seconds would generate approximately 200 messages. These values would become queued inside the slave and sequentially executed, which would result in an accumulating latency over a twenty second period. A method was required that would immediately send the last received position change when the last message was complete, but would discard previous values that were not yet actioned. We accomplished this through an independent thread for executing the post command. This thread would be effectively dormant while waiting for an event. When a message arrives on a different thread, the event is triggered, at which point the thread wakes and sends the message. We effected this through the use of Java synchronisation objects. The functionality that sends the post messages to Stellarium executes in an indefinite loop, laying dormant through the `altAzSynchroniser.wait()` call.

```
new Thread(() -> {
    while (!exitThread) {
        synchronized (altAzSynchroniser){
            try {
                altAzSynchroniser.wait();
            } catch (InterruptedException e)
                {
                e.printStackTrace();
            }
        }
        sendAltAz(currentAz, currentAlt);
    }
}).start();
```

The thread will wait indefinitely until it receives a signal from variable `altAzSynchroniser`. When a message to change the altitude arrives from the poi, the class variable `currentAlt` is set and the `altAzSynchroniser` object is notified, which in turn

[12] www.raspberrypi.org/documentation/linux/usage/rc-local.md [accessed November 2018]

[13] www.raspberrypi.org/forums/viewtopic.php?t=139224 [accessed November 2018]

[14] stellarium.org/doc/head/remoteControlApi.html

[15] curl.haxx.se

[16] cURL should not be confused with the curl programming language. ec.haxx.se/curl-name.html [accessed November 2018].

causes the thread shown above to wake and then call `sendAltAz` with the new azimuth and altitude to the localhost.

```
public void changeAltitude(double
    control_val) {
  synchronized (altAzSynchroniser){
   currentAlt = control_val  / 2 * Math.PI;
   altAzSynchroniser.notify();    }
}
```

We found that modifying the azimuth and altitude directly often produced a jittery display due to the 100ms latency coupled with discarding of values that were not actioned while waiting for the `sendPostMessage` call to return. We reduced this problem significantly by sending arrow key messages and moved the display left and right instead of sending an azimuth. This produced a smooth display rotation when rotating the ball. It was not possible to use this for the altitude in the terrestrial mode because we were using the accelerometer value to determine the height. In the spaceship mode, however, this proved very effective as we were able to just send up, down, left and right messages based on gyroscope action.

6. FUTURE WORK

There were several issues that we discovered through running the game. The first problem was that the Raspberry Pi would often crash when running the display after a certain period of intense manipulation, however, we were able to run it for several days if we did not demand too many rapid changes from Stellarium. We substituted the Pi with a Mac Mini in order to determine where the problems were. We found that we were able to reproduce an error in Stellarium on the Raspberry Pi when running the script *double_stars.ssc* that comes with Stellarium, however, the Mac ran with no errors. Running the kernel journal showed errors indicating an inability to allocate memory within the GPU[17]. The VC4 OpenGL driver required to run Stellarium is still experimental, and it is probably that this is where the error lies. Research and development in this area is still required to make a stable Raspberry Pi installation of Stellarium.

We found that when the player started rotating the ball fast, the zoom would activate, requiring them to stay within certain rotation rates. We modified the game so changing zoom required the player to hold the button down when performing a zoom action.

Messages are sometimes lost over UDP, which became evident when a zoom message was sometimes not delivered to the slave. We have performed some tests comparing different routers and different Raspberry Pis for packet loss. Additionally, we tested code in both Java and C++. We discovered that as packet intervals exceeded 10ms, the percentage of packet loss increased. Interestingly, we found that there was less packet loss using Java than C++ using the standard compilers distributed with Raspbian. Furthermore, the quality of router had a significant impact. Some routers, although supporting multicasting, stopped sending multicast messages to devices after about ten minutes. We intend to perform more tests regarding the packet loss, however, the real concern is that broadcasting and multicastling of OSC over UDP is not satisfactory [33].

We found that the Just In Time (JIT) compiler took time to convert the downloaded Java byte code into machine code [34], producing a brief stuttering effect when executed for the first time. The problem became exacerbated when using the Pi Zero with ten oscillators running simultaneously due to the limited power of the Pi Zero. Once the JIT compiler had converted the code, subsequent code changes were not affected. Although only an issue when the program starts, we need to examine strategies to overcome this.

[17] github.com/Stellarium/stellarium/issues/550 [accessed November 2018]

7. CONCLUSIONS

During our research we were able to integrate various open source programs to create a system where we could develop and evaluate Stellarium as a controllable display element, create inter process and device communication using the HappyBrackets Java environment, and to experiment with the use of the sonic poi as a performance tool. We used this system to create a gamified environment where visitors were engaged with our technology, providing them with a positive and memorable experience. We capitalised on this opportunity to observe and evaluate how our system was behaving, which was more memorable to us by virtue of it being part of a game that was played repeatedly. We leveraged the quality the Stellarium display coupled with a wireless control device to create a game that was challenging, fun, engaging and educational. Moreover, the technical goal was to be able to control Stellarium during a performance with HappyBrackets, with an example available at `https://youtu.be/NhXRdd-MNoo`

The research obtained from developing this game can be used as a starting point for the development of an interactive educational installation. Furthermore, we found a way to expose issues with OpenGL driver on the Raspberry Pi, Java JIT, and UDP packet loss and performance using both Java and C++.

8. ACKNOWLEDGEMENTS

A special thanks to Ben Cooper who designed and built the Sonic Poi. Many thanks to the members of the Newcastle Astronomical Society in their support and encouragement for this project. I would like to acknowledge the support of Georg Zotti and Alexander Wolf from the Stellarium development team for their advice and guidance in using Stellarium.

9. REFERENCES

[1] Sam Ferguson and Oliver Bown, "Creative coding for the Raspberry Pi using the HappyBrackets platform," in *Proceedings of the 2017 ACM SIGCHI Conference on Creativity and Cognition*. ACM, 2017, pp. 551–553.

[2] Eleanor Farrington, "Parametric equations at the circus: Trochoids and poi flowers," *The College Mathematics Journal*, vol. 46, no. 3, pp. 173–177, 2015.

[3] Karen Robson, Kirk Plangger, Jan H Kietzmann, Ian McCarthy, and Leyland Pitt, "Game on: Engaging customers and employees through gamification," *Business horizons*, vol. 59, no. 1, pp. 29–36, 2016.

[4] Nergiz Ercil Cagiltay, "Teaching software engineering by means of computer-game development: Challenges and opportunities," *British Journal of Educational Technology*, vol. 38, no. 3, pp. 405–415, 2007.

[5] Angelo Fraietta, "Musical composition with naked eye and binocular astronomy," in *Australasian Computer Music Conference 2014*. Victorian College of the Arts, 2014, p. 47.

[6] Angelo Fraietta, "Echoes from the fourth day - a segue through the southern night sky for FM synthesiser and binoculars," 2014, Performed in Brickworks Park in collaboration with the Newcastle Astronomical Society.

[7] Colin Bright, "spa-c–e," 2013, Performed live at Colbourne Ave Glebe, Sydney, Australia May 23rd 2013 by The Colin Bright Syzygy Band and Angelo Fraietta.

[8] Simon Monk, *Raspberry Pi cookbook: Software and hardware problems and solutions*, " O'Reilly Media, Inc.", 2016.

[9] Samuel Aaron, Alan F Blackwell, and Pamela Burnard, "The development of Sonic Pi and its use in educational partnerships: Co-creating pedagogies for learning computer programming," *Journal of Music, Technology & Education*, vol. 9, no. 1, pp. 75–94, 2016.

[10] Matt Richardson and Shawn Wallace, *Getting started with Raspberry Pi*, " O'Reilly Media, Inc.", 2012.

[11] Ivica Ico Bukvic, "Pd-l2ork Raspberry Pi toolkit as a comprehensive Arduino alternative in k-12 and production scenarios.," in *NIME*, 2014, pp. 163–166.

[12] Maik Schmidt, *Raspberry Pi: A Quick-Start Guide*, Pragmatic Bookshelf, 2014.

[13] Oliver Bown, Miriama Young, and Samuel Johnson, "A Java-based remote live coding system for controlling multiple Raspberry Pi units," in *ICMC*, 2013.

[14] Yusuf Abdullahi Badamasi, "The working principle of an Arduino," September 2014, pp. 1–4, IEEE.

[15] Oliver Bown, Lian Loke, Sam Ferguson, and Dagmar Reinhardt, "Distributed interactive audio devices: Creative strategies and audience responses to novel musical interaction scenarios," in *International Symposium on Electronic Art*. ISEA, 2015.

[16] Hicham Medromi, Laila Moussaid, and FAL Laila, "Analysis of the allocation of classes, threads and cpu used in embedded systems for Java applications," *Procedia computer science*, vol. 134, pp. 334–339, 2018.

[17] Joseph Ashley, "Computers and computer programs," in *Astrophotography on the Go*, pp. 151–161. Springer, 2015.

[18] K Berglund, "Using free, open source Stellarium software for iya2009," in *Preparing for the 2009 International Year of Astronomy: A Hands-On Symposium*, 2008, vol. 400, p. 483.

[19] Matthew Mc Cool, "Touring the cosmos through your computer: a guide to free desktop planetarium software," *CAPjournal,(7)*, pp. 21–23, 2009.

[20] Georg Zotti and Alexander Wolf, "Stellarium 0.18.0 user guide," 2018.

[21] Georg Zotti, Florian Schaukowitsch, and Michael Wimmer, "The skyscape planetarium," 2017.

[22] Patricia M Greenfield, Craig Brannon, and David Lohr, "Two-dimensional representation of movement through three-dimensional space: The role of video game expertise," *Journal of applied developmental psychology*, vol. 15, no. 1, pp. 87–103, 1994.

[23] Mark JP Wolf, "Genre and the video game," *The medium of the video game*, pp. 113–134, 2001.

[24] Robert J Stone, Peter B Panfilov, and Valentin E Shukshunov, "Evolution of aerospace simulation: From immersive virtual reality to serious games," in *Recent Advances in Space Technologies (RAST), 2011 5th International Conference on*. IEEE, 2011, pp. 655–662.

[25] Jukka Rönkkö, Jussi Markkanen, Raimo Launonen, Marinella Ferrino, Enrico Gaia, Valter Basso, Harshada Patel, Mirabelle D'Cruz, and Seppo Laukkanen, "Multimodal astronaut virtual training prototype," *International Journal of Human-Computer Studies*, vol. 64, no. 3, pp. 182–191, 2006.

[26] Jitong Chen, Lingquan Meng, Xiaonan Wang, and Chenhui Wang, "An integrated system for astronomical telescope based on Stellarium," in *Advanced Computer Control (ICACC), 2011 3rd International Conference on*. IEEE, 2011, pp. 431–434.

[27] Elena Tuveri, Samuel A Iacolina, Fabio Sorrentino, L Davide Spano, and Riccardo Scateni, "Controlling a planetarium software with a kinect or in a multi-touch table: a comparison," in *Proceedings of the Biannual Conference of the Italian Chapter of SIGCHI*. ACM, 2013, p. 6.

[28] Karen Robson, Kirk Plangger, Jan H Kietzmann, Ian McCarthy, and Leyland Pitt, "Is it all a game? Understanding the principles of gamification," *Business Horizons*, vol. 58, no. 4, pp. 411–420, 2015.

[29] Thomas W Malone, "Heuristics for designing enjoyable user interfaces: Lessons from computer games," in *Proceedings of the 1982 conference on Human factors in computing systems*. ACM, 1982, pp. 63–68.

[30] JW van der Veen, R de Beer, and D van Ormondt, "Utilizing Java concurrent programming, multi-processing and the Java native interface," *Running Native Code in Separate Parallel Processes," Report on behalf of the Marie-Curie Research Training Network FAST*, 2012.

[31] Cay S Horstmann and Gary Cornell, *Core Java 2: Volume I, Fundamentals*, Pearson Education, 2002.

[32] Matthew Wright, Adrian Freed, et al., "Open SoundControl: A new protocol for communicating with sound synthesizers.," in *ICMC*, 1997.

[33] Angelo Fraietta, "Open sound control: Constraints and limitations.," in *NIME*, 2008, pp. 19–23.

[34] Anderson Faustino Da Silva and Vitor Santos Costa, "An experimental evaluation of Java JIT technology.," *J. UCS*, vol. 11, no. 7, pp. 1291–1309, 2005.