Android Game Coursework: OpenWorld

Benjamin Howe
Department of Computer Science
The University of Reading, United Kingdom

Abstract – This report discusses the development of "OpenWorld", a sandbox Android game featuring procedurally generated worlds. Included is a flexible object-oriented model suitable for re-use in other games featuring players moving through worlds, as well as algorithms for procedurally generating worlds scattered with various resources.

Contents

To the coursework marker
Specific learning difficulties (such as dyslexia)
It is the recommendation of the University Specialist Teacher Assessor that this candidate's coursework should not be penalised for poor spelling, poor grammar or awkward sentence structure.

In your feedback, please state:
I have marked the candidate's work in accordance with this recommendation: Yes / No
If no, please indicate why: either
- It has been agreed that this module is exempt
- The recommendations are not relevant in terms of the content of the work (e.g. the work is numerical)

To the student
This notice should only be used if you have been authorised to do so. Misuse by other students may result in disciplinary action.

RA number: RA20140462

1 Introduction & Showcase

This report details the development of a procedurally generated sandbox Android game, from very basic "template" code to final product. The object oriented design, performance and memory usage, and procedural world generation are discussed in-depth in later sections.

The game was inspired by similar games (both on Android and other platforms), such as Terraria[1] and Minecraft[2]. It is designed with a pixelated and blocky appearance, and can be installed from a small APK (less than 150KB). The object orientated design means that much of the code written can be imported into other similar projects.

Figure 1: The finished game with the character in the default position. The seed was "UoR".

Figure 2: The finished game with the character in near the bottom of the map. Two diamond deposits are visible. A zombie can be seen falling into lava (where it will die due to excessive lava damage). Beneath the lava, bedrock can be seen. The seed was "UoR".

[1] https://terraria.org/
[2] https://minecraft.net/

2 Object Orientated Design

2.1 Introduction

This game was designed to exploit the efficiencies that object oriented programming offers, by making extensive use of inheritance for the different objects within the game. For example, the Player class inherits a number of basic methods (relating to movement) from Lifeform, which inherits a basic method for returning a bitmap to be drawn to the screen from Entity.

As this game was written for Android (as opposed to desktop devices) various special considerations had to be made. For example, Android devices can have a wide variety of display characteristics. A Samsung Galaxy S1 has a 4 inch screen at 233 PPI [1], and a Samsung Galaxy S6 has a 5.1 inch screen at 577 PPI [2]. The game should run on both devices – but clearly assets would have to be at much higher resolution to look crisp on the larger screen of the S6.

The aim of this project was learn and demonstrate knowledge of Java applied to Android programming, by creating a procedurally generated sandbox game.

2.2 Method

The base code consisted of four classes – GameThread, GameView, MainActivity, and TheGame. Only limited object orientation was exhibited (the ball, paddle, and face "targets" were not objects). Initially, the code was stripped back to remove the sensors from GameView and to everything (the ball, paddle, and targets) from TheGame. Then, a basic player object was created, instantiated within TheGame, and drawn to the screen. The next step was to animate the player when the right and left edges of the screen were touched – this represented movement[3] and was done by creating an "update" function to be called immediately before drawing each frame.

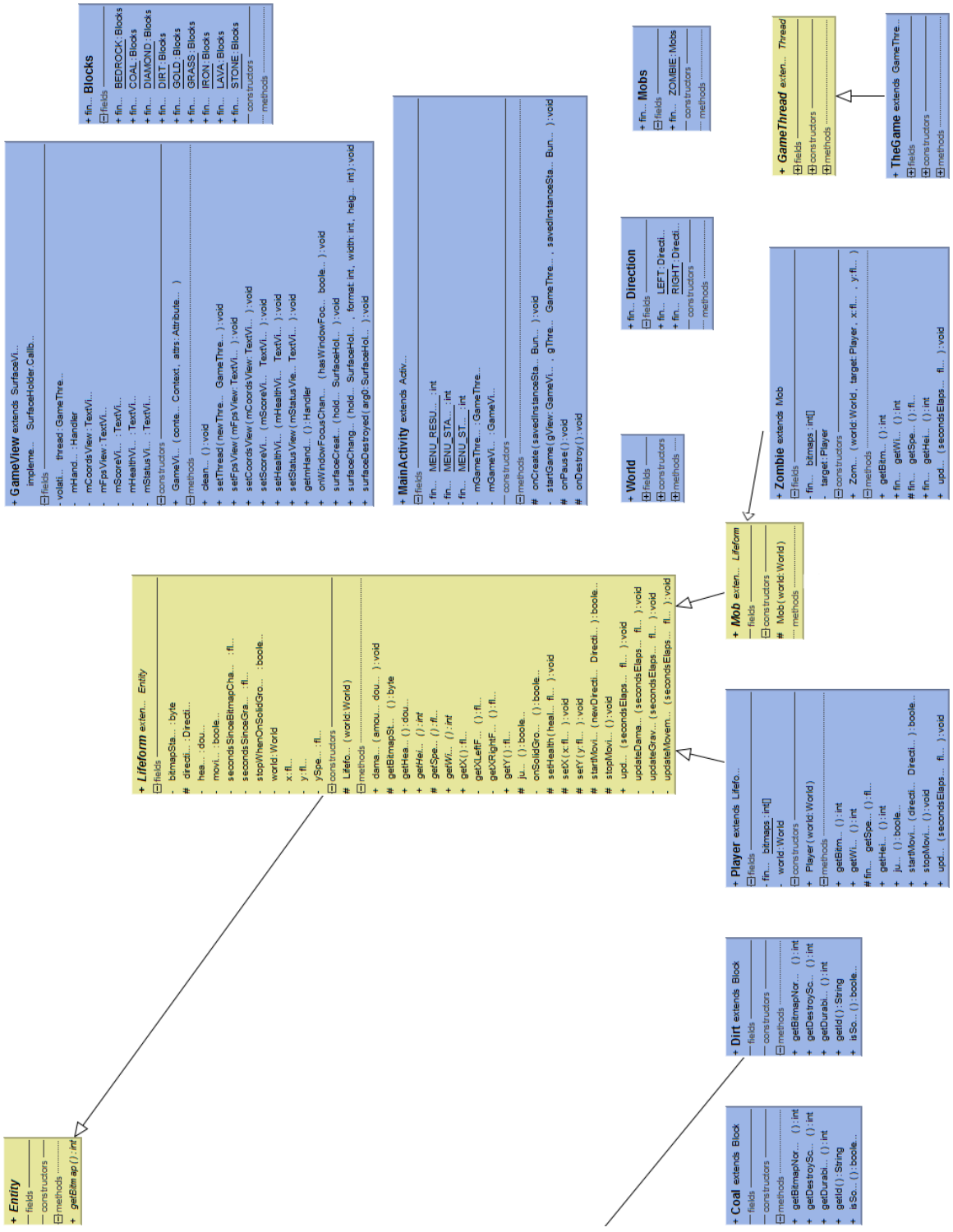
After this, a basic world class was created for the player to exist within – and a pointer to this world was added to the player. The world was updated through the player – the players update function called the world update function. Blocks were added to the world, as objects stored in an array. Initially the world was filled with dirt blocks, but as game development progressed more complex structures were added through specific methods for generating veins and clusters of blocks (for more details see later chapter on procedural generation).

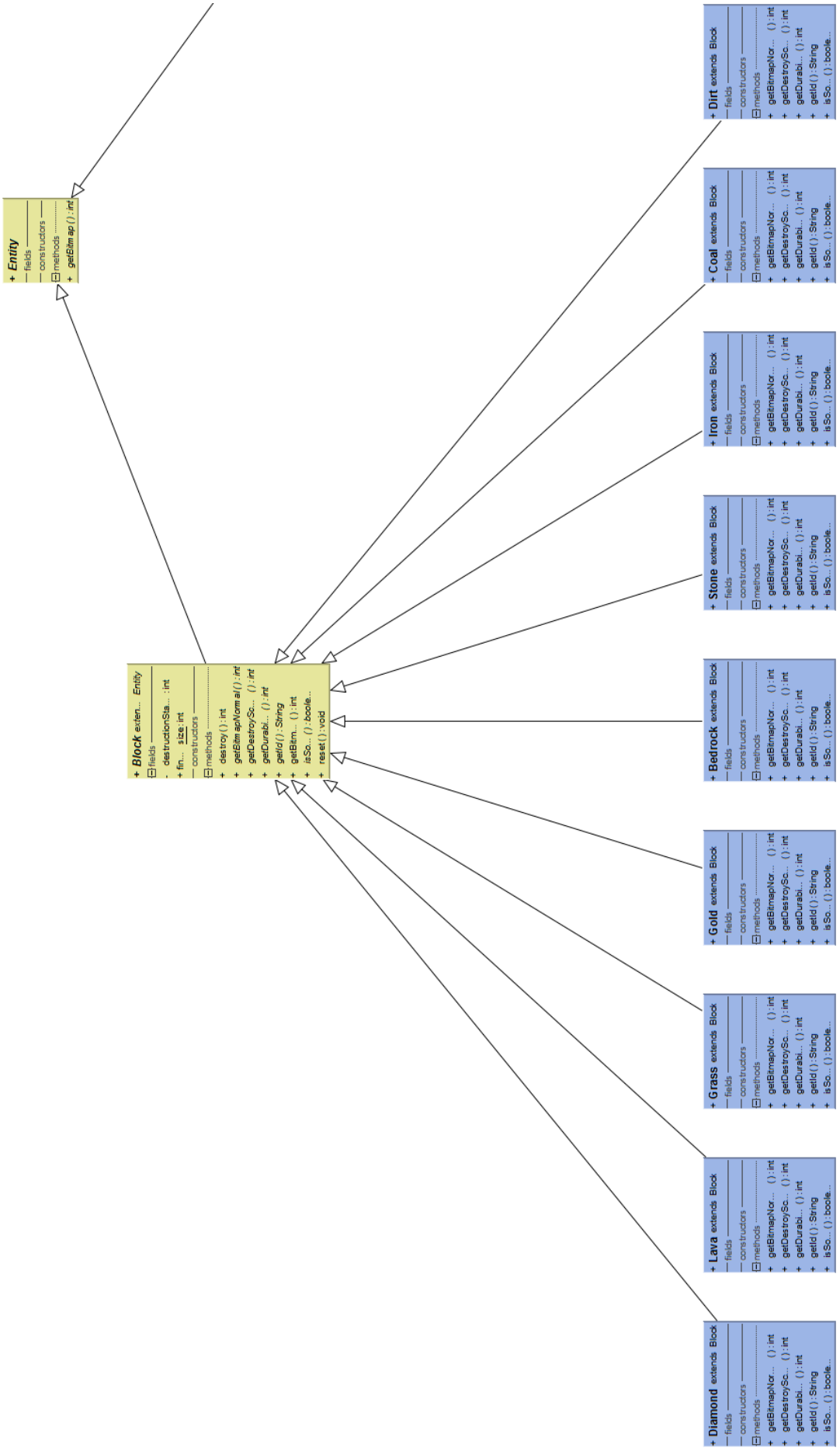
Much of the player functionality was then moved into the parent class, Lifeform. This was so that mobs (zombies, and potentially other mobs – e.g. goblins, dwarves…) could inherit some of the basic properties of the player that are common to all lifeforms (e.g. falling due to gravity, movement…). Logcat [3] was regularly used to assist debugging and to allow early code to be written without graphical output.

All the graphical assets (the background [4], the player sprites [5], and the zombie sprites [6]) except the blocks were taken from a variety of third party sources. The blocks were designed using Paint.NET[4] as 14 x 14 pixel blocks, and then upscaled using the nearest neighbour algorithm in order to maintain crispness. Layouts were designed within Android Studio (based on IntelliJ) and defined in XML. Strings were stored as resources, where possible, to aid potential future language translation [7].

[3] Movement is actually achieved by moving the background and blocks equally and oppositely to the movement of the player – i.e. when the player appears to be moving left this is because the background is moving right.
[4] https://www.getpaint.net/

2.3 Result


+ GameView extends SurfaceVi...
+ fin... Blocks
+ fin... Mobs
+ GameThread exten... Thread
+ TheGame extends GameThre...
+ MainActivity extends Activ...
+ fin... Direction
+ World
+ Zombie extends Mob
+ Lifeform exten... Entity
+ Mob exten... Lifeform
+ Player extends Lifefo...
+ Entity
+ Dirt extends Block
+ Coal extends Block


Entity
Block
Dirt
Coal
Iron
Stone
Bedrock
Gold
Grass
Lava
Diamond

2.4 Discussion

The design made good use of the features of object oriented programming, such as inheritance. This is in contrast to the base code, which appears to only use objects where required by Android, and prefer to use procedural programming.

Objects were updated using ticks (i.e. the player moved 0.2 blocks every 0.1 seconds). This may have been suboptimal – arguably it would have been better to have moved the player 2 blocks every second, but to do this by moving the player a fraction of 2 blocks every time the update function was called, depending on what fraction of a second had passed.

The base code appears to have been originally developed by a Perl developer and / or appears to be heavily based on Google's Lunar Lander sample from 2008 [8] because of the consistent prefixing of variables with "m" (assumed to be short for "my" – a keyword in Perl used to declare a variable as local as opposed to global [9]). New variables were not prefixed with "m" – as it appears to be a redundant character that adds no value. However, where existing code was modified (e.g. the addition of mHealthView in the GameView class) the "m" was retained for consistency.

2.5 Conclusions

This Android game was designed by following best practices, both for object oriented programming (e.g. by making use of inheritance) and Android development (e.g. by storing strings as resources as opposed to embedding them within the Java code). The design favours practicality over theoretical correctness (e.g. the circular reference between player and world), allowing the game to be robust while enabling rapid development.

3 Memory Usage & Speed Improvements

3.1 Introduction

Android is a mobile operating system, typically installed on mobile phones or tablets (although it is increasing in popularity on televisions / set-top boxes and in vehicles). In order to prolong battery life, mobile devices typically feature lower power (and therefore slower) CPUs compared to desktop devices. They also use a different CPU architecture; the vast majority (over 98%) of Android devices use ARMv7 (RISC) chips as opposed to Intel x86 (CISC) chips [10]. Compared with iOS devices (and even some desktop computers), Android devices allow for more simultaneous threads of execution – while the majority of iOS devices (almost 95%) are only single or dual threaded [11], 82.7% of Android devices support 4 or 8 simultaneous threads [10].

The app, unoptimised, used approximately 16MB of RAM and was drawn at approximately 10 FPS on an Nvidia Shield tablet[5]. As the RAM usage was not of concern (as over 80% of Android devices feature 1GB of RAM or more [12]), the main aim of this experiment was to improve the framerate.

3.2 Method

Significantly more time is spend drawing to the screen than is spent computing physics and updating the game, therefore this area was focussed on for performance improvements. The bitmap assets were processed in four different ways – "control" (no compression, no scaling), "compressed" (using PunyPNG Pro[6]), "scaled" (using a custom Python script and the Pillow library [13]), and "scaled and compressed".

All of these experiments were run on the aforementioned Nvidia Shield tablet – and for all tests (where applicable) the seed used was "UoR". Prior to each build and launch the build was manually cleaned – as if this wasn't done then the APK could grow by up to 10KB (which is not normally significant, but as this application is so small it can result in an APK that is 17% larger than necessary!).

The size of the APK was measured, as was the memory usage and FPS when static and falling / digging (via adb). The static test was run until the player was killed by a generated zombie, and the falling / digging test was run until the player hit y = 100 (i.e. descended 28 blocks).

3.3 Result

Tests involving the CPU and GPU (i.e. those recording FPS) were run multiple times, as these are contended resources. Tests involving RAM and storage usage were only run once, as the results should always be the same.

	APK Size (KB)	
	Value	**% better**
Scaled & Compressed	60.5	-3.77
Scaled	61.7	-5.83
Compressed	54	7.38
Control	58.3	n/a

Table 1: Showing APK size vs the compression and / or scaling of bitmap assets.

[5] Nvidia Tegra K1 SoC, 4x ARM Cortex A-15 @ 2.22 Ghz, GeForce ULP K1 (192 cores @ 950 MHz) GPU, 2GB RAM, 32GB eMMC, Android 7.0 (build NRD90M.1928188_805.6612)
[6] http://www.punypng.com/pro

	Avg. FPS					RAM Use (MB)	
	1	**2**	**3**	**Avg.**	**% better**	**Avg.**	**% better**
Scaled & Compressed	17.85	17.89	17.79	17.84	27.89	15.5	0.64
Scaled	17.94	17.93	17.92	17.93	28.53	15.6	0.00
Compressed	14.12	14.04	14.00	14.05	0.57	15.5	0.64
Control	14.31	14.24	14.29	13.95	n/a	15.6	n/a

Table 2: Showing average FPS and RAM usage vs the compression and / or scaling of bitmap assets while the player is static.

	Avg. FPS					RAM Use (MB)	
	1	**2**	**3**	**Avg.**	**% better**	**Avg.**	**% better**
Scaled & Compressed	10.23	10.28	10.40	10.30	30.55	15.5	0.64
Scaled	10.30	10.22	10.21	10.24	29.78	15.6	0.00
Compressed	7.66	7.64	7.40	7.57	-4.06	15.5	0.64
Control	7.88	7.96	7.82	7.89	n/a	15.6	n/a

Table 3: Showing average FPS and RAM usage vs the compression and / or scaling of bitmap assets while the player is falling / digging.

3.4 Discussion

Perhaps unsurprisingly, compressed images use less space in the APK file. Having images at multiple scales increases the amount of space used by just under 6% - although the APK was still only ~60KB so arguably this is irrelevant.

As expected, it was found that the game uses approximately 16MB of RAM at all times. This is an acceptable level of memory usage. Memory usage decreased fractionally when using compressed images – possibly because the source images were smaller.

Significant improvements were made to the framerate by scaling the images to the correct sizes. This suggests that Android doesn't cache scaled drawable objects – which seems odd, as the screen size of an Android device usually remains the same. Compression of the source images made very little difference to framerate – this is because the zlib decompression algorithm (deflate) that PNG images use [14] is affected very little by compression level – sometimes more heavily compressed images are actually quicker to decompress because they're smaller to read from storage.

3.5 Conclusions

The performance of this app was boosted by approximately 30% by pre-scaling the images. Performance was increased by a further 30% after exporting the APK as "release". Although this is still significantly below 60 FPS (the framerate recommended by Google [15]) most of the animation is footsteps, and therefore some choppiness is acceptable.

4 Improvements and Extensions – Procedural World Generation

4.1 Introduction

Figure 3: an example of a rock structure procedurally generated in a Minecraft world.

Procedural generation in video games is making a comeback. In the early 1980s, procedural generation was used to allow games to feature near-infinite content using extremely limited amounts of code (e.g. the entirety of Rouge – a popular 1980s procedurally generated dungeon game where no two games were ever the same – was just 360KB of C [16]). Later games, for example RoboBlitz, used tools such as ProFX to procedurally generate textures in order to reduce the size of the game to only 5% of the original footprint [17]. In the modern-day, Minecraft used procedural generation (Perlin noise) to create infinite worlds containing elaborate and unexpected details [18].

4.2 Method

In order to control the world generator, it was important that the user could manually set the seed – in Java, the random number generator is seeded with a long [19]. However, longs aren't very memorable, so it was decided to set the seed as a string, and then to hash the string into a long. The code used for this was very similar to String.hashCode() – except it returned a long, as opposed to an int [20].

If the seed was blank, then the current time in milliseconds since midnight, January 1, 1970 UTC was used instead.

Once the seed was generated, the seeded random number generator was used to create a world: firstly the "normal" blocks were created (dirt at the top of the map, gradually transitioning into stone. Then, ores were added using specific functions to insert veins (for coal) and clusters (for iron, gold, and diamond), with different frequencies and quantities occurring at different levels. Finally, lava was added at y = 1, and bedrock at y = 0, x = 0, and x = 1023. By generating a world this way, identical worlds were created when the random number generator was seeded with identical seeds.

4.3 Result

```
long worldSeed = 1125899906842597L; // prime
Log.d("Seed", "Seed = \"" + seed + "\".");
int len = seed.length();
if (len > 0) {
    Log.d("Seed", "Seed is NOT blank - hashing.");
    for (int i = 0; i < len; i++) {
        worldSeed = 31*worldSeed + seed.charAt(i);
    }
    Log.d("Seed", "Seed hash = \"" + worldSeed + "\".");
} else {
    Log.d("Seed", "Seed is blank, using currentTimeMillis().");
    worldSeed = System.currentTimeMillis();
}
float screenWidthInBlocks = mCanvasWidth / (Block.size *
displayDensity);
float screenHeightInBlocks = mCanvasHeight / (Block.size *
displayDensity);
world = new World(worldSeed, screenWidthInBlocks, screenHeightInBlocks);
```

Figure 5: the code used for seeding the random number generator used to create the world. Note that the string "seed" contains the string that the user entered in the box on the menu screen to be used at the seed (so could be blank). This code is used in the setupBeginning() function in the class "TheGame".

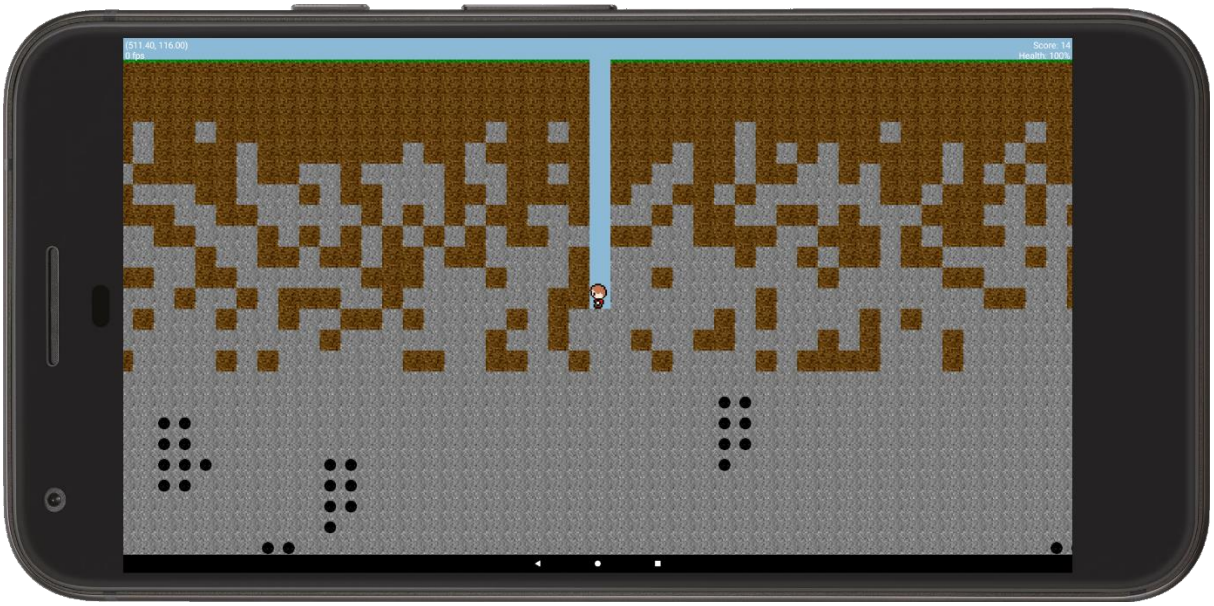


Figure 4: an example of a procedurally generated world, using the seed "UoR".

4.4 Discussion

The use of procedural generation in this game means that every session will be unique (unless two sessions are started at the exact same millisecond, or the seed is manually set from the menu). The other obvious advantage is that levels do not need to be designed.

There were many extra features that could be added. The ground in the game world was flat – it could have had terrain by using 1-dimensional Perlin noise [21]. Game worlds were finite – they measured 1024 blocks by 256 blocks. While restricting the height of the world probably made sense (both in a practical sense and for storage purposes), it would be perfectly feasible to store chunks of (measuring 1024 by 256) in an ArrayList, and then to generate the world

from a random number generator with a fixed seed (i.e. before generating a chunk, set the seed back to the seed used to generate the world, then change it in some way specific to that chunk) – this would ensure that two worlds started with the same seed would always be identical, even if the chunks are generated in different orders.

4.5 Conclusions

Some progress was made towards creating a completely procedurally generated world. The code used worked robustly, and handled both "lazy" users (who did not want to manually set a seed) and more advanced users who wanted to play the same map as a friend, or who wanted to play on a map they've played on before. More work could be done, though, for example terrain generation and facilitating truly unlimited world – although perhaps the latter was unnecessary for this coursework as a world 1024 blocks wide would take over 8 minutes to walk from end-to-end nonstop.

5 Bibliography

[1] GSMArena, “Samsung I9000 Galaxy S,” March 2010. [Online]. Available: http://www.gsmarena.com/samsung_i9000_galaxy_s-3115.php. [Accessed 23 April 2017].

[2] GSMArena, “Samsung Galaxy S6,” March 2015. [Online]. Available: http://www.gsmarena.com/samsung_galaxy_s6-6849.php. [Accessed 23 April 2017].

[3] Google, Inc, “Write and View Logs with Logcat,” November 2016. [Online]. Available: https://developer.android.com/studio/debug/am-logcat.html. [Accessed 23 April 2017].

[4] Pantone, “PANTONE 14-4318 TCX,” 2013. [Online]. Available: http://www.pantone.com/color-finder/14-4318-TCX. [Accessed 23 April 2017].

[5] isaiah658, “Retro Character Sprite Sheet,” Open Clipart, 6 May 2016. [Online]. Available: https://openclipart.org/detail/248259/retro-character-sprite-sheet. [Accessed 23 April 2017].

[6] Kazzador and Enterbrain, “SpriteSpace: VX Fantasy Sprites,” BlogSpot, 1 October 2013. [Online]. Available: http://spritespace.blogspot.co.uk/2013/10/to-start-of-with-here-are-collection-of.html. [Accessed 23 April 2017].

[7] Google, Inc, “String Resources,” May 2010. [Online]. Available: https://developer.android.com/guide/topics/resources/string-resource.html. [Accessed 23 April 2017].

[8] The Android Open Source Project, “samples/LunarLander/src/com/example/android/lunarlander - platform/development - Git at Google,” 21 October 2008. [Online]. Available: https://android.googlesource.com/platform/development/+/master/samples/LunarLander/src/com/example/android/lunarlander. [Accessed 23 April 2017].

[9] J. Allen, “my - perldoc.perl.org,” May 2016. [Online]. Available: https://perldoc.perl.org/functions/my.html. [Accessed 23 April 2017].

[10] Unity Technolgies, “Mobile (Android) Hardware Stats 2017-03,” March 2017. [Online]. Available: https://hwstats.unity3d.com/mobile/cpu-android.html. [Accessed 23 April 2017].

[11] Unity Technologies, “Mobile (iOS) Hardware Stats 2017-03,” March 2017. [Online]. Available: https://hwstats.unity3d.com/mobile/cpu-ios.html. [Accessed 23 April 2017].

[12] Unity Technologies, “Mobile (Android) Hardware Stats 2017-03,” March 2017. [Online]. Available: https://hwstats.unity3d.com/mobile/mem-android.html. [Accessed 17 April 2017].

[13] B. Howe, “BenjaminEHowe/resize-for-android.py,” GitHub, 23 April 2017. [Online]. Available: https://gist.github.com/BenjaminEHowe/80da06fec5da10364280333cc473f807. [Accessed 23 April 2017].

[14] W3C, “Portable Network Graphics (PNG) Specification (Second Edition),” 10 November 2003. [Online]. Available: http://www.libpng.org/pub/png/spec/iso/index-object.html#10Compression. [Accessed 23 April 2017].

[15] Google Developers, “Android Performance Patterns: Why 60fps?,” YouTube, 6 January 2015. [Online]. Available: https://www.youtube.com/watch?v=CaMTIgxCSqU. [Accessed 23 April 2017].

[16] D. Silva, “Davidslv/rogue,” GitHub, 23 July 2016. [Online]. Available: https://github.com/Davidslv/rogue/. [Accessed 23 April 2017].

[17] J. Orry, “RoboBlitz gets clever texturing,” VideoGamer.com, 6 March 2006. [Online]. Available: https://www.videogamer.com/news/roboblitz-gets-clever-texturing. [Accessed 23 April 2017].

[18] J. Fingas, “Here's how "Minecraft" creates its gigantic worlds,” Engadget, 3 April 2015. [Online]. Available: https://www.engadget.com/2015/03/04/how-minecraft-worlds-are-made/. [Accessed 23 April 2017].

[19] Oracle, “Random (Java Platform SE 8),” 2016. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/Random.html#Random-long-. [Accessed 23 April 2017].

[20] sfussenegger, “What is a good 64bit hash function in Java for textual strings?,” Stack Overflow, 3 June 2010. [Online]. Available: http://stackoverflow.com/a/1660613. [Accessed 23 April 2017].

[21] H. Elias, “Perlin Noise,” February 1999. [Online]. Available: https://web-beta.archive.org/web/20160530124230/http://freespace.virgin.net/hugo.elias/models/m_perlin.htm. [Accessed 23 April 2017].

[22] K. Perlin, “An Image Synthesizer,” in *SIGGRAPH '85 Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 1985.

A Glossary of Key Terms, Acronyms, and Abbreviations

APK	Android PacKage: a file containing all the information necessary to install and run a piece of software on Android devices.
ARMv7	A 32-bit ARM architecture, commonly used in mobile and low power devices.
CISC	Complex Instruction Set Computing: a CPU design strategy where instruction sets are complex, so bespoke circuitry can be designed to perform complex tasks in hardware. Common CISC architectures include x86 and AMD64. The opposite of RISC.
FPS	Frames Per Second.
iOS	The operating system for Apple's mobile devices.
Perlin noise	A type of noise developed by Ken Perlin in 1983, used to produce textures of natural appearance [22].
PPI	Pixels per inch: a measure of display density.
RISC	Reduced Instruction Set Computing: a CPU design strategy where instruction sets are simple, and complex tasks require multiple instructions. Common RISC architectures include MIPS and ARM. The opposite of CISC.