UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Anton Tšugunov

Shuriken Way – An Android Puzzle Game

Bachelor's thesis (9 EAP)

Supervisor:
Raimond-Hendrik Tunnel, MSc

Tartu 2018

Shuriken Way – An Android Puzzle Game

Abstract:

The thesis describes the development and testing of an Android video game called Shuriken Way, which was developed as a game that provides a unique player experience. The game was developed without the use of game engines. The choice of different technologies is explained and some alternative routes the development could have taken are analyzed. The thesis details the implemented game mechanics and the design of prominent game levels. Finally, the testing stage of the work is described. The game was tested on multiple Android devices to discover compatibility issues. The game was then compared against other similar Android games found on the Google Play distribution platform to see how Shuriken Way compares to them in terms of performance. Lastly, the game was playtested with new players to uncover any issues that players can face when playing the game. Based on the results of the tests, improvements were made or proposed for the future developement of the game.

Keywords:

Game, game development, game design, puzzle game, mobile, 3D, 2.5D, graphics, physics-based puzzles, VBO, OpenGL, Android

CERCS: P170 Computer science, numerical analysis, systems, control

Shuriken Way – mõistatusmäng Android-seadmetele

Lühikokkuvõte:

Antud bakalaurusetöös kirjeldatakse Android-mängu arendamist ja testimist. Mängu nimeks on Shuriken Way. Mängu arendati mänguna, mis pakub mängijale ainulaadset mängukogemust. Mängu arendamiseks ei kasutatud mängumootoreid. Erinevate tehnoloogiate valik on töös põhjendatud ja alternatiivsed lähenemised on analüüsitud. Töös on detailselt kirjeldatud mängus rakendatud mängumehaanikaid ning väljapaistvamate mängutasemete disaini. Töö lõpus on kirjeldatud mängu testimine. Mängu testiti mitmes erinevas Android-seadmes, et avastada ühilduvusprobleeme. Seejärel on töös võrreldud mängu jõudlust teiste Google Play levitusplatvormist leitud sarnaste Android-mängudega. Lõpuks anti mäng uutele mängijatele mängimiseks, et selgitada välja probleemid, millega mängijad võivad mängimisel kokku puutuda. Testimise tulemuste põhjal arendati mängu edasi ning töös on toodud välja ettepanekud mängu arendamiseks tulevikus.

Võtmesõnad:

Mäng, mängude arendamine, mängu disain, mõistatusmäng, mobiilne, 3D, 2.5D, graafika, füüsika-põhised mõistatused, VBO, OpenGL, Android

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

1 Introduction

There are a lot of mobile games on the Android (see Appendix I for definition) application distribution platform called Google Play[1] but not all of them are original ideas brought to life. Many of them are just "clones" of previously existing popular games. [1] Present thesis describes the development and testing of an Android video game *Shuriken Way*. The aim was to develop *Shuriken Way* as a game that provides a unique player experience instead of trying to mimic the experience one could already get from existing games. In addition, *Shuriken Way* needed to perform well and have great entertainment value. *Shuriken Way* is a physics-based puzzle game, where the goal of the player is to complete all the levels while getting as high of a score as possible in each of them. The player is given control of a shuriken (see Appendix I for definition) and the goal of every level is to collect all the coins in it. This is usually done by hitting the coins with the shuriken. Very often reaching a coin requires the player to plan out their actions and use certain mechanics of the surrounding objects. Throughout the levels there are also objects that are explicitly there to try to stop the player from reaching some coins. The score for a level is based on how quickly the player was able to complete it.

There are other publicly available games for Android that are similar to *Shuriken Way* but there is a number of aspects that make *Shuriken Way* comparatively unique. Chapter 2 of the thesis gives an overview of similar games available on Google Play to date and compares them to *Shuriken Way*.

The game was developed with Android Java and OpenGL ES (Open Graphics Library for Embedded Systems) without using game engines. Chapter 3 of the thesis explains the choice regarding those and other technologies that were used to develop the game and lists some alternative routes the development could have taken. Advantages and disadvantages of each route are analyzed.

Chapter 4 of the thesis describes the implemented game mechanics for each game object and how each object's mechanics are or can be used in puzzles. Chapter 5 describes how prominent game levels were designed. The chapter explains the main aspects of them that make the levels "puzzling" and how different game objects and mechanics were used to achieve the result.

[1] https://play.google.com/store

The game was tested on multiple devices to discover performance, visual or any other compatibility problems. The game was then further developed to improve or fix any device compatibility issues found during the testing. In addition, *Shuriken Way* was tested alongside other similar games found on the Google Play to see how it compares to the alternatives in terms of graphics performance.

Finally, *Shuriken Way* was playtested by new players to uncover any confusion or problems that players can face when playing the game. This includes level imbalance, game breaking bugs, unintuitive controls and bad game mechanics. Improvements were made or planned for the future development. The whole testing stage is detailed in the chapter 6 of the thesis.

Some terms used in the present thesis are defined in the Glossary (Appendix I). An installation guide for the game is available in Appendix II. A brief game guide is available in Appendix III. The game installation package (*shuriken-way.apk*), a short video demonstration of the gameplay (*demo.mp4*) and the source code files (*/Source*) are available in the Accompanying Files (Appendix IV). The following Illustrations 1, 2, 3 and 4 are screenshots of *Shuriken Way* taken at the end of the work for the present thesis.

Illustration 1. Level select menu of *Shuriken Way*.

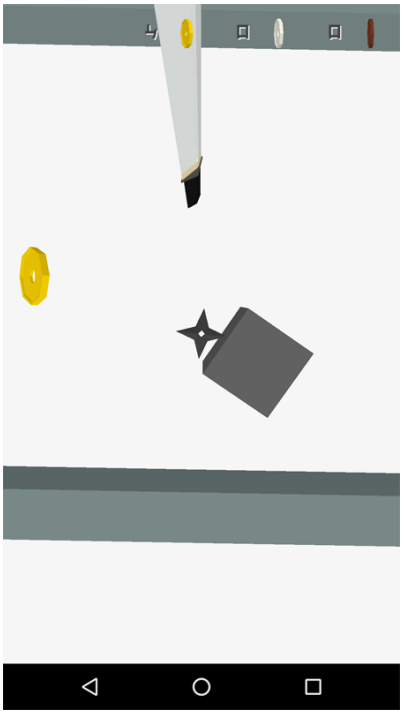

Illustration 2. Screenshot taken during the level 12 of *Shuriken Way*.

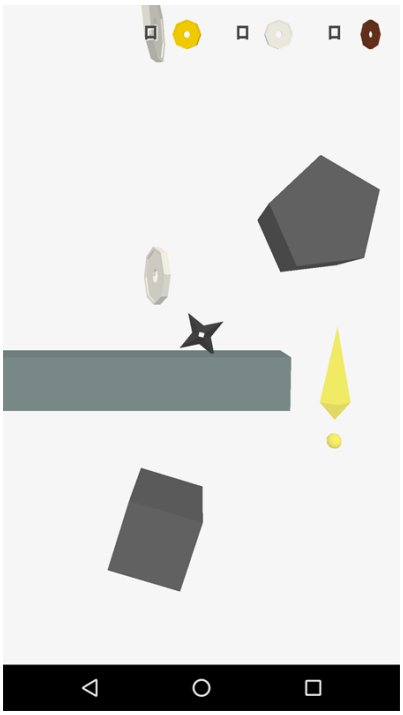

Illustration 3. Screenshot taken during the level 5.

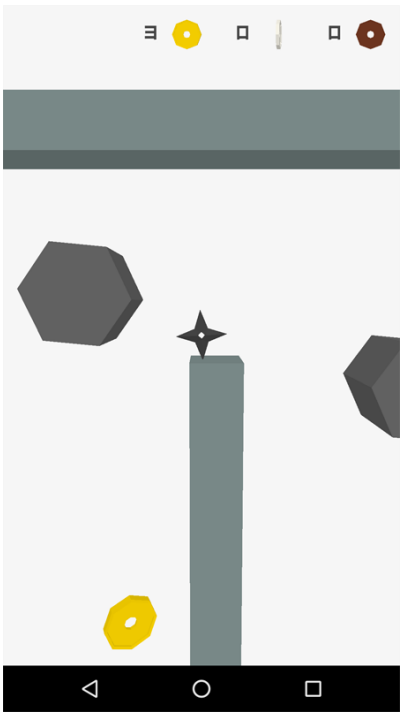

Illustration 4. Screenshot taken during the level 8.

2 Alternatives

As mentioned in the Introduction, *Shuriken Way* was developed during the work for the present thesis as a game that provides a unique player experience. However, this fact does not prohibit or negate the existence of some visual similarities or similar mechanics between *Shuriken Way* and existing games. As long as the actions that the player has to take in order to see the visuals or trigger a mechanic are different, it is possible to create a unique experience. Like J. Schell [7:143] has written: "The actions a player can take are so crucial to defining a game's mechanics that changing a single action can give you a completely different game." Two Android games have been found that are similar to *Shuriken Way* enough to make a comparison: *Ninja Star!* and *Shuriken*. Subchapters 2.1–2.2 go in more detail on the two games.

2.1 Ninja Star!

Ninja Star! (Illustration 5) is a video game by Piggyback Games[2] developed for Android and iOS operating systems. It was released in the spring of 2015. The game has 1000–5000 downloads on Google Play to date[3].

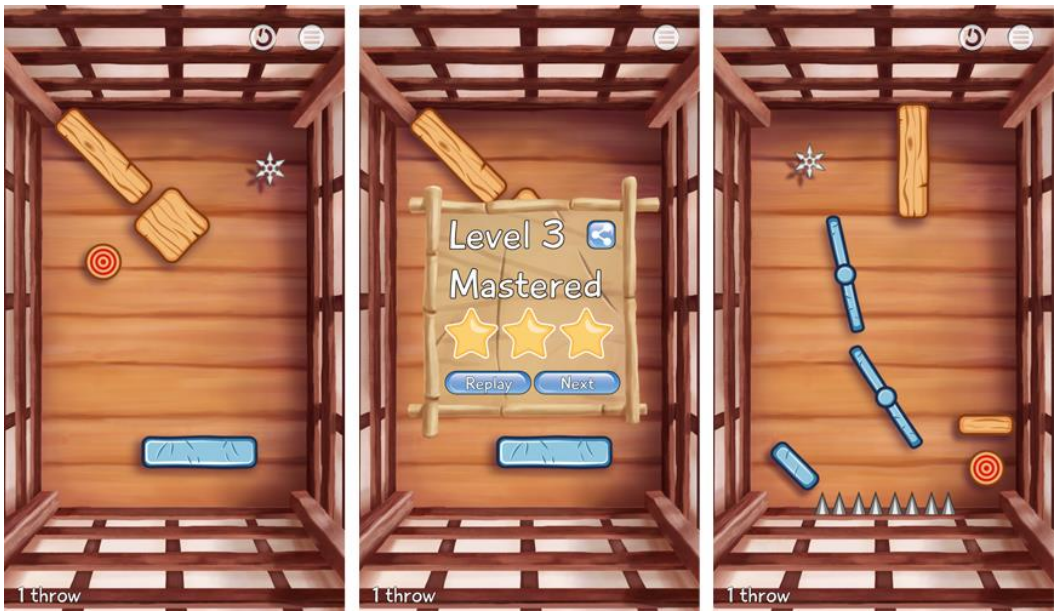

Illustration 5. Screenshots of the game *Ninja Star!*

The most notable similarities between *Ninja Star!* and *Shuriken Way* are in the overall theme and genre. *Ninja Star!* is also a multi-level physics-based puzzle game involving a shuriken.

[2] http://www.piggybackgames.com/

[3] https://play.google.com/store/apps/details?id=com.piggyback.ninjagolf

The games also share a number of mechanics that can be considered identical but these similarities in mechanics come mostly from simulating real world physics, for example bouncing and collision. Despite the similarities, there are a lot of differences between the two games which make *Shuriken Way* stand out and make it unique.

The most "visible" difference is exactly the visual aspect: *Shuriken Way* uses 3D (or sometimes called 2.5D) graphics with minimal texture use, whereas *Ninja Star!* chose a heavily texture-based 2D approach (see Appendix V). Just as an important of a difference is the performance which, despite of *Ninja Star!* using 2D graphics, is arguably worse in *Ninja Star!* than *Shuriken Way*. The differences in performance of *Shuriken Way* and the alternatives are documented in detail in subchapter 6.2 of the present thesis.

Lastly, there are a lot of differences in game mechanics. *Ninja Star!* uses a fixed camera (i.e. view) and the size of the level is limited to the area visible by the camera, whereas *Shuriken Way* uses a camera that is locked onto the controlled shuriken and the size of the level is only limited by the finiteness of the Android Java variable types and the mobile device capabilities. In *Ninja Star!* the player chooses where to throw the shuriken by touching the screen in the wanted direction, which requires touch precision. In *Shuriken Way*, the direction in which the shuriken is being thrown is determined by the rotation of the object that the shuriken is stuck in. This emphasizes precise timing from the player, thus creating a reaction-based challenge. In addition, *Shuriken Way* contains game objects with mechanics that are not present in *Ninja Star!*, for example rotating platforms that cause the camera and the direction of the gravity to rotate alongside the platforms when the shuriken is stuck in them and rotating landable walls. These objects are described in more detail in the chapter 4 of the present thesis.

2.2 Shuriken

Shuriken is a video game for Android which was last updated in June of 2016. The exact release date of the game is unknown. It was uploaded to Google Play by the user *DnL*. The game has 500–1000 downloads on Google Play to date[4]. Just like *Ninja Star!*, *Shuriken* shares a similar theme with *Shuriken Way* but the genre is different (Illustration 6 and 7).

[4] https://play.google.com/store/apps/details?id=com.DnL.Shuriken

Illustration 6. Screenshot of *Shuriken*.

Illustration 7. Screenshot of *Shuriken*.

Shuriken is visually very similar to *Flappy Bird*, a mobile game that got very popular in 2014 and shares similar mechanics.

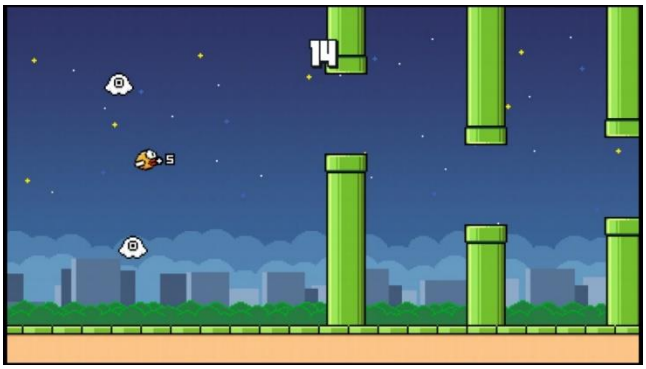

Illustration 8. Screenshot of the game *Flappy Bird*.

Flappy Bird is a game where the player is given control of a bird. The aim is to keep the bird in the air and avoid the incoming pipes by tapping the screen to fly up when necessary (Illustration 8). It is very similar to how the player avoids logs in *Shuriken*.

Apart from the theme, there are not any noteworthy similarities between *Shuriken* and *Shuriken Way*. The games differ in graphics, goal, engine etc. *Shuriken* uses texture-based 2D graphics, compared to the 3D low texture graphics of *Shuriken Way*. *Shuriken* does not consist of multiple levels and instead uses an infinite level approach similar to *Flappy Bird*.

Shuriken was developed using the Unity game engine compared to the Android Java and OpenGL ES approach of *Shuriken Way*. The choice of not using game engines like Unity for the development of *Shuriken Way* is explained in the next chapter of the thesis.

3 Technologies Used

As mentioned in the previous chapter, *Shuriken Way* is developed using the Android Java programming language and the built-in OpenGL ES 2.0 API (Application Programming Interface). The IDE (Integrated Development Environment) used for the development is the Android Studio 3.0.1[5]. Rendering is handled using VAO/VBO (Vertex Array Object / Vertex Buffer Object) and GLSL (OpenGL Shading Language) shaders technology directly. That makes the development different from using game engines, which provide an abstraction layer between the game content and the underlying hardware [3].

Such approach was chosen over using game engines like Unity[6] and Unreal Engine[7] for many reasons, including educational, and is further explained in subchapter 3.5. The following subchapters 3.1–3.4 provide a brief overview of the technologies used in the development of *Shuriken Way*.

3.1 Android Java

The Android Java (i.e. Java for Android) is a programming language that is almost identical to the regular Java language. The Android Java largely uses Java's adoption of the object-oriented programming model and is designed to work with respect to the regular Java's concepts. There are differences though, one of which is the compilation process. Regular Java code is compiled to its bytecode form and then run on a Java Virtual Machine[8] instance. The Android Java code is also compiled to its bytecode form but, starting from Android version 4.4, it is also converted to device-based machine code when installing the compiled application to a device. This means that the application does not run on a virtual machine. [5] The Android Java was used in the development of *Shuriken Way* to program game mechanics and to access OpenGL ES for rendering graphics.

3.2 OpenGL ES

Android uses OpenGL ES as an API to graphics hardware. OpenGL ES consists of a set of commands and functions that let the programmer specify programs, objects and operations

[5] https://developer.android.com/studio/index.html

[6] https://unity3d.com

[7] https://www.unrealengine.com

[8] https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-1.html#jvms-1.2

involved in rendering high-quality graphics, specifically colour images of 3D objects. Typical programs that use OpenGL ES open a window into a framebuffer, allocate an OpenGL ES context and associate it with the window. Next, commands to define shaders, textures, geometry can be made as well as commands to draw the defined geometry. [8] In the case of *Shuriken Way*, OpenGL ES version 2.0 is used. The version 2.0 has been chosen over versions 1.0, 1.1, 3.0 and 3.1. Both 2.0 and 3.* provide faster graphics performance than the older versions but newer features of the versions 3.* were not needed for the development of *Shuriken Way*. OpenGL ES 3.0 and 3.1 have better support for texture compression than version 2.0 but it is not necessary in the case of *Shuriken Way* because its graphics use very few textures, mainly for text rendering.[9] OpenGL ES 2.0 is also supported by a wider variety of devices than the newer versions, which is why it was currently preferred[10]. The geometry in *Shuriken Way* is defined by the use of VAO and VBO technology built into the OpenGL ES API.

3.3 VAO and VBO

VAO or vertex array objects represent a collection of vertex attribute sets. For the definition of vertex, see the Glossary (Appendix I). These attribute sets are stored in the buffer object data store. [8] If a buffer object contains vertex data, for example spatial coordinates, normal vectors etc, it is commonly called a vertex buffer object or VBO for short[11]. In the case of *Shuriken Way*, the technology is used as intended, to predefine and store vertex data of the game object models on the graphics hardware so that shader programs can access it during rendering.

3.4 GLSL Shaders

Shaders are pieces of code that can be compiled for and executed on the GPU. The shaders in OpenGL are written in The OpenGL Shading Language (GLSL). Different GLSL shaders can be written for each of the programmable steps in the OpenGL processing pipeline: vertex, tessellation control, tessellation evaluation, geometry, fragment, and compute. A collection of compiled shaders written for different steps is called a shader program. [9] In the case of *Shuriken Way*, the required shaders for vertex and fragment steps have been

[9] https://developer.android.com/guide/topics/graphics/opengl.html#choosing-version

[10] https://developer.android.com/guide/topics/graphics/opengl.html#version-check

[11] https://www.khronos.org/opengl/wiki/Vertex_Specification#Vertex_Buffer_Object

written. Writing other types of shaders is optional and only useful when it is needed to customize other stages of the processing pipeline. Default operations in those stages are performed otherwise[12]. The vertex step operates on the vertices and the data associated with them [9]. The fragment step, on the other hand, is responsible for operations on fragments and the data associated with them, for example depth values [9]. For the definition of fragment, see the Glossary (Appendix I). At the end of the pipeline, an image representing the current state of the game can be seen on the screen and the player can respond to it accordingly. The next subchapter examines some alternative routes the development of *Shuriken Way* could have taken and explains the choice of discarding them.

3.5 Alternative Routes

Shuriken Way was developed without the help of game engines for a number of reasons. The main reason why the chosen approach to the game development was preferred is for the purpose of gaining more experience in using low level techniques for rendering computer graphics. The use of the two most popular games engines[13] was still considered and it is analyzed in the following subchapters 3.5.1 and 3.5.2.

3.5.1 Unity

Illustration 9. Logo of the Unity engine.

The biggest advantage of Unity (Illustration 9) is the ability to efficiently support a wide variety of mobile devices. This includes optimized cross-platform shaders. [2] With Unity the number of compatibility issues with different types of devices can be drastically lowered. Unity was not chosen for the development of *Shuriken Way* for the following reasons. Using Unity the game developers have less control over the lower levels of the engine which means that there may be bugs in the game engine which cannot be fixed until the developers of Unity fix the issue and release an update. Another disadvantage of using Unity in the context of the present thesis is author's lack of experience with Unity, which would have slowed down the development of *Shuriken Way*. The most important disadvantage of Unity is the high consumption of CPU,

[12] https://www.khronos.org/opengl/wiki/Vertex_Processing

[13] https://www.gamedesigning.org/career/video-game-engines/

GPU, energy and memory resources [3]. Prior experience of the author of the present thesis is that Android games developed with Unity have very often underperformed Android games developed using alternative methods. This is supported by the small preliminary study conducted for the present thesis. The description of the study and the results are shown in the Appendix VI.

3.5.2 Unreal Engine 4

Just like Unity, Unreal Engine 4 (Illustration 10) efficiently supports a wide variety of mobile and static devices [4]. However, as mentioned before, game engines consume a lot of device CPU, GPU, energy and memory resources [3]. This is once again one of the reasons why Unreal Engine 4 was not chosen in this thesis. *Shuriken Way* was not planned as a very voluminous game and most of the Unreal Engine features were not required as they are there to speed up the development of massive graphically intensive games. The aim was for *Shuriken Way* to have very good performance to compete with similar games on the market. Taking performance into consideration, minimal experience with the engine would slow down the development of *Shuriken Way* compared to the Android Java and OpenGL ES approach that the author is more used to. All of these factors played roles in the decision to discard Unreal Engine 4. The next chapter provides an overview of the main game mechanics of *Shuriken Way* and details the way they have been implemented.

Illustration 10. Logo of Unreal Engine.

4 Game Mechanics

Game mechanics are the rules and operations of a game. They also describe the goal of the game and help the player to or hinder them from achieving the goal. Mechanics define the game [7]. *Shuriken Way* can be described as a game where the shuriken is interacting with other objects encountered in the levels. Several game objects with various mechanics have been designed and implemented for the present thesis. The following subchapters 4.1–4.5 go into detail about the implemented objects and the mechanics associated with them.

4.1 The Shuriken

The present subchapter does not provide a historical or technical overview of the Japanese throwing weapons known as shuriken. This is purely about the *shuriken* as the main game object of *Shuriken Way*. However, the shuriken object is inspired by the mentioned Japanese weapon and its visual appearance in the game mimics that of the real life object. The model was made with modeling tools in the free 3D computer graphics software called Blender[14] (Illustration 11).

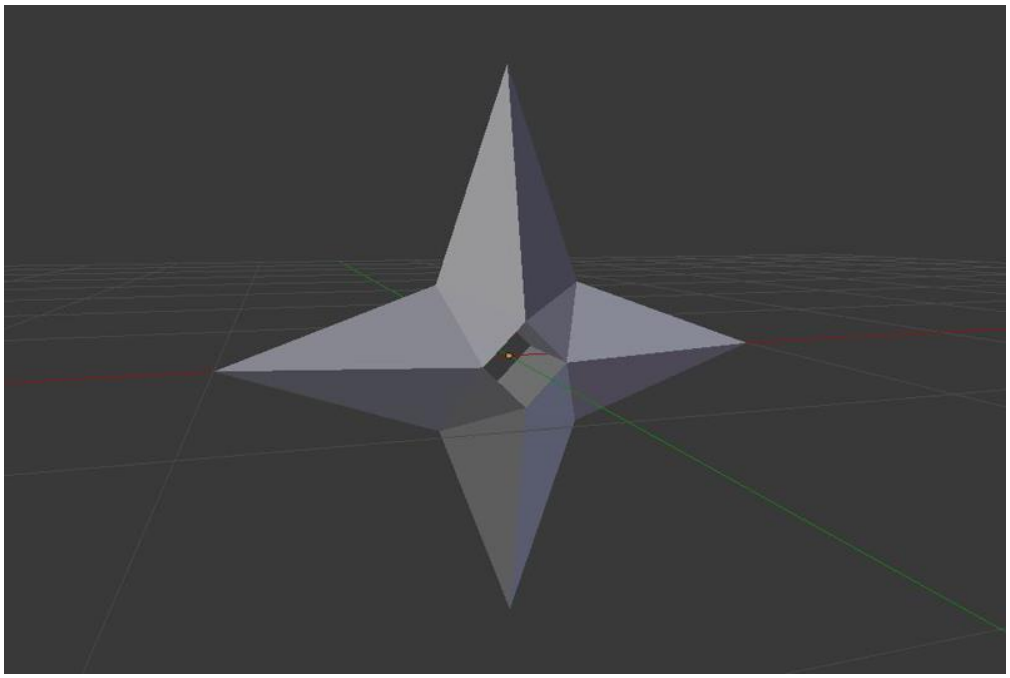

Illustration 11. The 3D model of the shuriken created using Blender.

The shuriken is the object that the player is given the control of. This is the most important mechanic of *Shuriken Way*. The shuriken takes part in most of the mechanics related to other

[14] https://www.blender.org/

object types, for example the housing mechanic of the platforms and the collection mechanic of the coin. The shuriken object's own main mechanics are the gravity mechanic that pulls the shuriken in a certain direction and the jumping or throwing mechanic which lets the player move the shuriken in the level. The direction that the gravity mechanic pulls the shuriken in will be called the direction of gravity further in the thesis. There is also another mechanic, which is that the level will end in a loss if the shuriken has moved above a level-specific threshold value in the direction of gravity (Illustration 12).

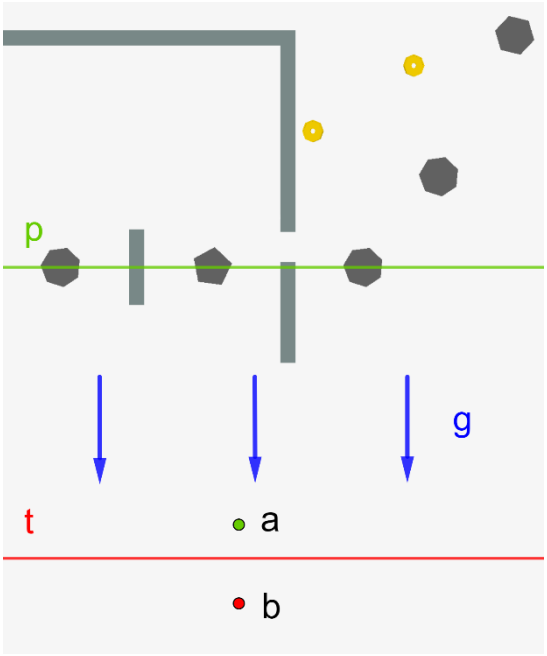

Illustration 12. Threshold value for the fall in the direction of gravity (*g*) visualised with the line *t*. Line *p* represents zero on the gravity axis. By moving from point *a* to point *b* the shuriken would cross the threshold.

This mechanic will be sometimes called "falling into the void" further in the thesis. The following subchapter 4.2 describes the different types of platforms that the shuriken object can interact with.

4.2 Platforms

Platforms are the core of *Shuriken Way*. Jumping from one platform to another is the main method used by the players to move around the current level. The platforms can rotate around their own axis perpendicular to the level. That rotation is what the player uses to aim their next jump. The platforms can be scaled according to the needs of a particular level or puzzle. Four types of platforms have been implemented in *Shuriken Way* during the work

for the present thesis. Subchapters 4.2.3–4.2.6 detail the platform type-specific mechanics. The following subchapters 4.2.1–4.2.2 describe the housing and collision detection mechanics that all types of platforms have in common.

4.2.1 The Housing Mechanic

In *Shuriken Way*, being stuck to an object is called being *housed* by the object. All platforms can house the game's main object – the shuriken. This means that any transformation applied to the housing platform will also apply transformation to the shuriken so that it looks relatively natural to the player. Transformations consist of moving and rotating the platform. In other words, the following happens when the shuriken collides with a platform:

1. The rotation and position (relative to the platform) of the shuriken are saved.
2. The rotation of the platform is saved.
3. The shuriken is marked as housed by the platform.

And while being marked as housed, the following is true for the shuriken:

1. The shuriken is moved relatively to the housing platform. The relative position is the one saved on collision but rotated according to the platform's rotation since the collision (Illustration 13).
2. The shuriken's rotation is changed according to the platform's rotation since the collision.

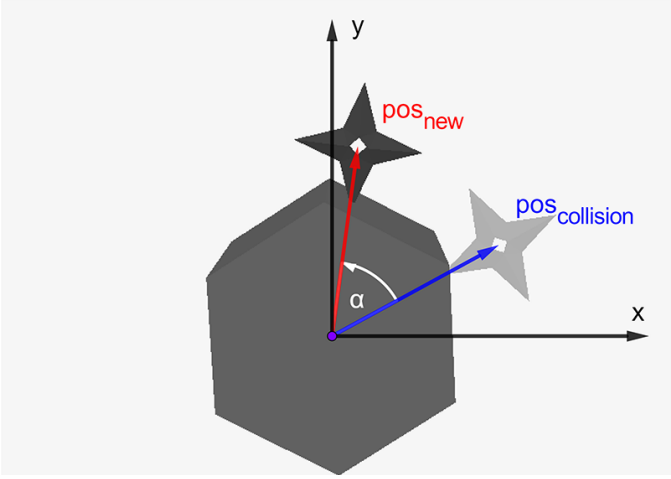

Illustration 13. The relative position of the shuriken rotating with the housing platform. Vector $pos_{collision}$ is the relative position of the shuriken on collision. Vector pos_{new} is the relative position of the shuriken while being housed. Vector pos_{new} is $pos_{collision}$ rotated by α, which is the platform's rotation since collision. Vectors x and y represent the horizontal (x-)axis and the vertical (y-)axis.

The player can detach the shuriken from the housing platform by tapping the screen. Collision detection, used for housing, and the mechanic of detaching the shuriken are partly or fully specific to the type of the platform. The following subchapter 4.2.2 describes the collision detection elements common to all types of platforms.

4.2.2 Collision Detection

Collision detection is calculating whether a game object intersects with another game object [10]. It is used by the platforms in *Shuriken Way* to determine when a platform needs to start housing the shuriken. Collision detection is done by performing operations on several points that have been virtually placed on the shuriken (Illustration 14). These points will be called *collision points* further in the text.

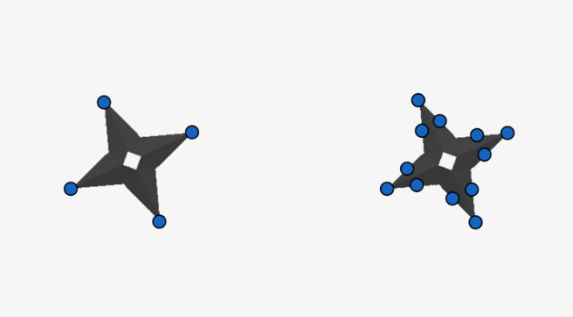

Illustration 14. A total of 4 collision points
placed on the left shuriken in the illustration and
12 collision points on the right one.

If one of the collision points fits the platform type-specific criteria for a collision, the collision is considered detected. The more collision points there are the more realistic the collision detection is. A total of 4 collision points are used in the version of *Shuriken Way* developed during the work for the present thesis. Every platform and the shuriken have bounding circles. The bounding circle determines how close a collision point needs to be to the object's center for the collision to be possible. There are no vertices of the object outside its bounding circle. If the platform's bounding circle does not intersect with the shuriken's bounding circle, then it is safe to say that there is no collision without performing the operations on the collision points (Illustration 15). The bounding circles of two objects intersect when the distance between their center points is less than the sum of the radiuses of the bounding circles.

Illustration 15. The bounding circles of the shuriken and a platform intersecting on the left and not intersecting on the right of the illustration.

4.2.3 The Simple Platform

The *simple platform* is the most common type of an object in *Shuriken Way*. It is in the shape of a right regular prism[15] and the bases of the prism can be regular polygons[16] with 4–8 sides.

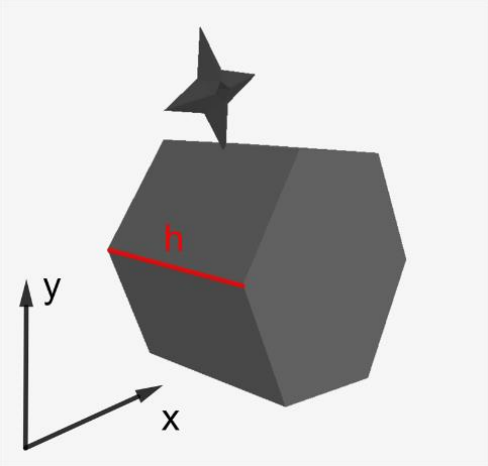

Illustration 16. The 3D model of a simple 6-sided platform housing the shuriken. Vectors *x* and *y* represent the horizontal (*x*-)axis and the vertical (*y*-)axis. Line segment *h* is the height of the right regular prism model.

Mechanic-wise the simple platform is considered here just a regular polygon. The height of the prism does not play a role in the platform's mechanics (Illustration 16). It is there purely for a visual effect.

[15] http://www.mathwords.com/r/right_regular_prism.htm

[16] http://www.mathwords.com/r/regular_polygon.htm

4.2.3.1 Collision

Collision detection between the simple platform and a collision point is based on the angle at which the point is relative to the platform's center point, 0 degrees being the positive direction of the horizontal axis (*x*, right), 90 degrees being the positive direction of the vertical axis (*y*, up). For simplicity, this angle will further be called a collision point's angle. Knowing the angles of all platform side normals, the normal (angle) closest to the collision point's angle can be found. The absolute value of the difference between the collision point's angle and the closest normal angle is the angular distance to the closest normal. Based on the angular distance, the maximum linear distance at which the collision point is still inside the platform can be calculated using trigonometry. If the collision point is at least as close as the calculated maximum distance, then the collision criteria is met. Illustration 17 shows an example of collision detection for a simple platform with 6 sides.

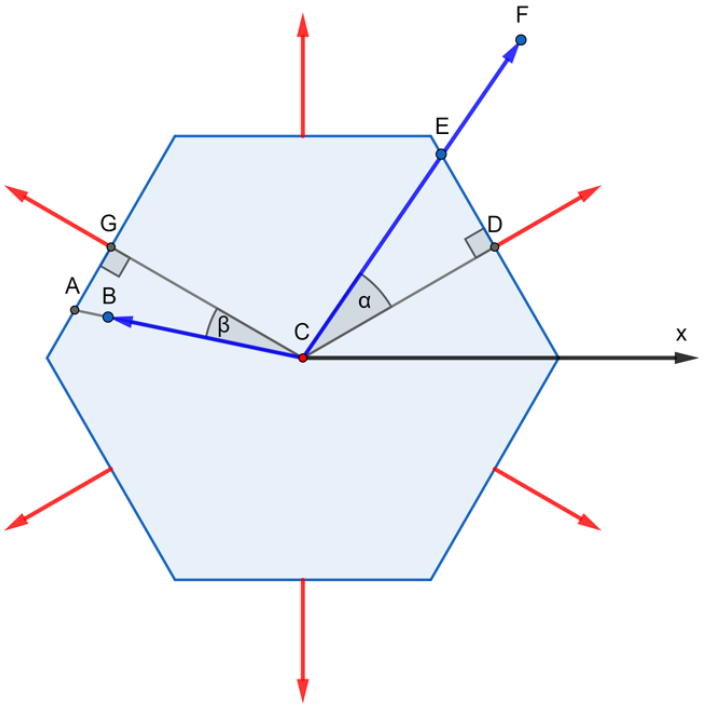

Illustration 17. Collision detection for a simple platform with 6 sides. Vector *x* represents the horizontal (*x*-)axis.

In the illustration, *F* and *B* are 2 examples of collision points, where *F* does not meet the criteria for collision and *B* does. *CD* and *GC* are equal line segments with a known length. Red vectors represent the normals of the platform sides. Angles α and β are angular distances

to the closest normals for collision points *F* and *B* respectively. Lengths of *EC* and *AC* are the maximum distances for collision that can be calculated using trigonometry.

To implement the described principle, a function for finding the angular distance to the closest normal has been written. There is no need to compare the collision point's angle to every platform side normal to find the closest normal. This is due to the platform's landable sides forming a regular polygon mechanic-wise. Moreover, every platform has a point that is directly to the right (direction of *x-axis*) of the platform's center.

A collision point's angle and the corresponding closest normal's angle difference is always in the range of $[-b, b)$ degrees, where b is equal to 180 divided by the number of sides that the platform has (Illustration 18).

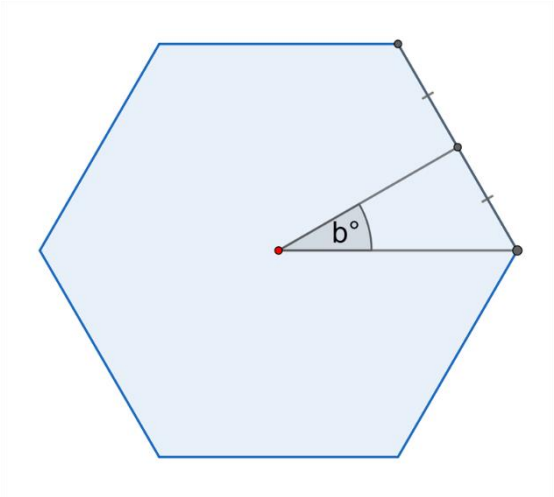

Illustration 18. Visual definition of the value b for a simple platform with 6 sides. In this example, b is 30.

The function for finding the angular distance to the closest normal can be presented as the following mathematical formula:

$$f(a, 2b) - b \tag{1}$$

Notation a in the formula (1) represents the collision point's angle. Function f in the formula is defined as the following:

$$f(x, y) = \begin{cases} x \,\%\, y, & x > 0 \\ x \,\%\, y + y, & otherwise \end{cases} \tag{2}$$

The symbol % in formula (2) stands for the division remainder operation. The function call $f(a, 2b)$ returns a value in the range $[0, 2b)$. Subtracting b in the formula (1) gives us the range $[-b, b)$. For a non-rotated platform, a collision point at 0 degrees and a collision point at $2b$ degrees will both have the angular distance to the closest normal of $|-b| = b$ degrees. A collision point at b degrees will have the angular distance to the closest normal of 0. To account for the platform's rotation, the platform's clockwise rotation is added to the collision point's angle (counterclockwise) before calculating the angular distance.

4.2.3.2 Detaching

Detaching from the simple platform sends the shuriken flying away from the platform's center. When the player taps the screen, acceleration in the needed direction is applied to the shuriken and it is unmarked from being housed by the platform.

4.2.3.3 Usage in Puzzles

Although the simple platform is the most common object type in the game, the puzzle aspect of it is not very significant. However, there are multiple ways that the player can land on a platform. Where and when the player succeeds to land affects the speed at which the player is able to move across the level. It is often crucial when going for the best score in the level.

4.2.4 The Relativity Platform

The *relativity platform* is very similar to the simple platform. Visually it is the same as the simple platform with the exception of being coloured differently. It has the same collision detection and detaching mechanics as the simple platform. However, the housing mechanic is significantly different. While housing the shuriken, the relativity platform applies the same transformations to the shuriken as every other platform but it also applies transformations to the level itself. While the platform is housing the shuriken, the rotation of the platform is also applied to the camera's angle around the *z*-axis and the angle of the gravity vector. The *z*-axis is the same axis that the platform is rotating around, which is perpendicular to the plane that the shuriken is moving on. This creates an effect of everything in the level rotating with both the platform and the shuriken staying in place (Illustration 19).

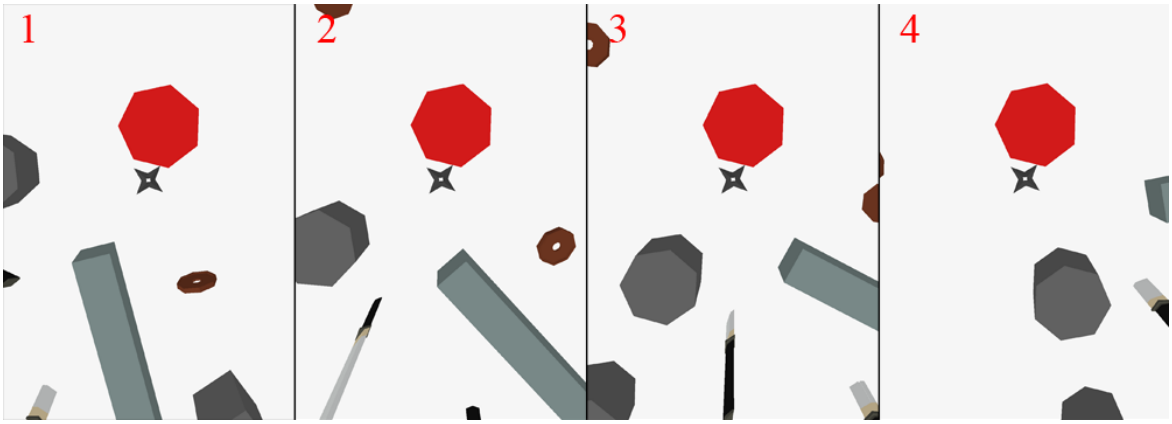

Illustration 19. The relativity platform applying transformations to the level.

Even after the shuriken has detached from the platform, the transformations applied to the level remain.

4.2.4.1 Usage in Puzzles

The puzzle aspect of the relativity platform is essential in *Shuriken Way*. The relativity platform makes it possible to create levels that are not bound to a single direction of gravity. The least obvious use for the relativity platform is placing it in a level as a decoy. Landing on the decoy platform and continuing with the level might not lead to a successful completion of the level. In such cases, the player needs to think of a way to avoid landing on the platform, which is sometimes not easy to do. More frequently, the player needs to instead use the mechanics of the relativity platform to progress further in the level. One of the reasons might be that some other important game objects, for example coins or platforms, may be more difficult or impossible to reach with the shuriken without changing the direction of gravity.

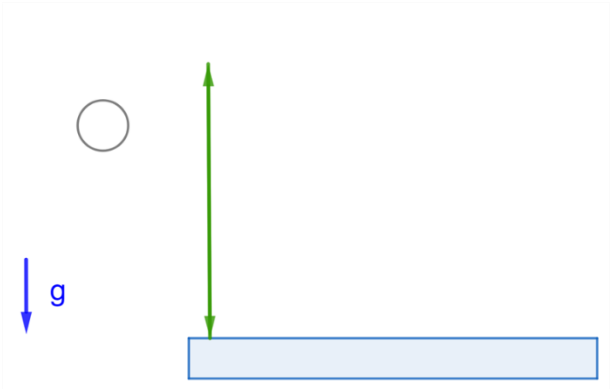

Illustration 20. A coin represented by a circle is not reachable from the rectangular wall because the gravity direction *g* is downwards. The green arrow represents the trajectory of the shuriken.

For example, there might be a simple wall in the level which is perpendicular to the direction of gravity and the shuriken has landed on the side of the wall opposite to the direction of gravity. The shuriken needs to collect a coin which is not reachable from the wall by jumping directly against the gravity (Illustration 20).

However, as a result of the relativity platform mechanics, the wall can change its rotation relative to the direction of gravity. This will affect the trajectory that the shuriken would follow after detaching from the wall. The previously unreachable coin can become reachable (Illustration 21).

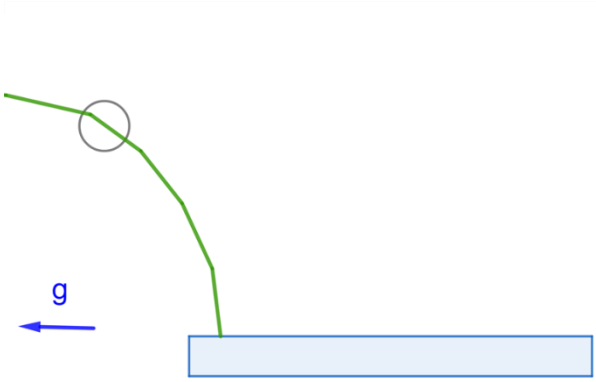

Illustration 21. A coin represented by a circle is reachable from the rectangular wall because the gravity direction *g* is to the left. The green line represents the trajectory of the shuriken.

The mechanics of the simple wall related to this are described in the following subchapter.

4.2.5 The Simple Wall

The *simple wall* is the second most common object type in *Shuriken Way*. Illustration 22 shows the simple wall it is seen in the game.

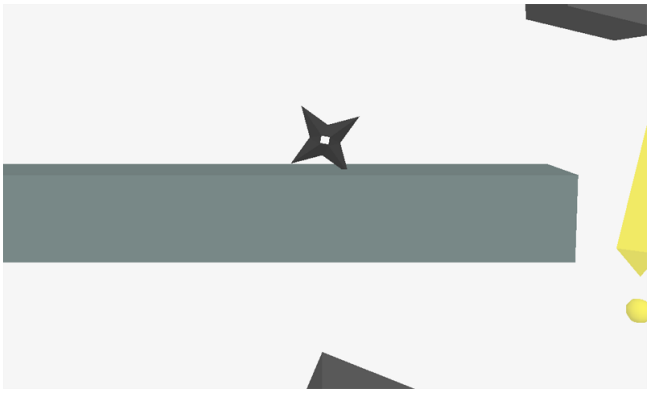

Illustration 22. A simple wall object housing the shuriken.

Visually, the simple wall is in the shape of a rectangular parallelepiped[17]. Mechanic-wise, the simple wall can be and is considered a rectangle.

4.2.5.1 Collision

To detect collision between the simple wall and a collision point, both the collision point's position relative to the wall's center and the wall itself are rotated clockwise by the counterclockwise rotation of the wall. This aligns the wall's width and height with the horizontal (*x*-) and vertical (*y*-)axises. Doing calculations is only required for rotating the relative position of the collision point. Illustration 23 shows an example of 4 collision points being rotated.

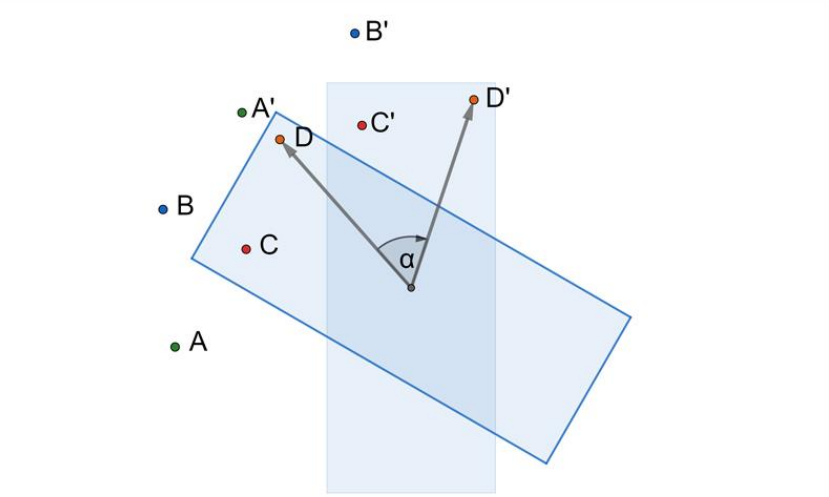

Illustration 23. The blue bordered rectangle represents a simple wall. Collision points *A*, *B*, *C*, *D* are rotated clockwise by the angle *α*, which counters the rotation of the wall. *A'*, *B'*, *C'* and *D'* are the collision points after the rotation.

The bounds of the wall after the rotation are simply its width and height, which are known values. Simple comparisons determine whether the rotated collision point is inside the wall's bounds. Because both the collision point and the wall are rotated equally around the same point, the collision detection also works for the state before the rotation.

4.2.5.2 Detaching

Detaching from the simple wall works slightly different from the simple and the relativity platforms. Instead of sending the shuriken flying from the center point of the platform, the

[17] http://www.mathwords.com/r/rectangular_parallelepiped.htm

shuriken is sent flying in the direction of the normal of the wall side that the shuriken is stuck to. The side that the shuriken is on is determined by a simple algorithm. Terms “above”, “below” and “left of” will be further used to describe the property of having a higher *y* coordinate, a lower *y* coordinate and a lower *x* coordinate than something else respectively. The coordinates of the shuriken’s center relative to the wall’s center and the wall itself are rotated the same way it is done for collision detection (Illustration 23). If in this state the shuriken’s center is above or below the wall bounds, then the landed side is determined as the top or the bottom respectively. Otherwise it is checked whether the shuriken’s center is to the left of the wall bounds. If it is, then the landed side is determined as left and determined as right otherwise. In a non-rotated state the normal vectors (*x*, *y*) of the right, top, left and bottom wall sides are (1, 0), (0, 1), (-1, 0) and (0, -1) respectively. The normal coordinates of the determined wall side are rotated by the wall rotation and used as the direction for the jump.

4.2.5.3 Usage in Puzzles

The simple wall is usually used as a barrier to the shuriken. It gives the player no other choice but to find and follow another path around the wall. However, the simple wall can also sometimes be used by the player to speed up the level completion. For example, instead of waiting for a rotating platform to fully rotate in the wanted direction, the player can jump to the side of a wall that is facing the wanted direction and then jump off the wall (Illustration 24).

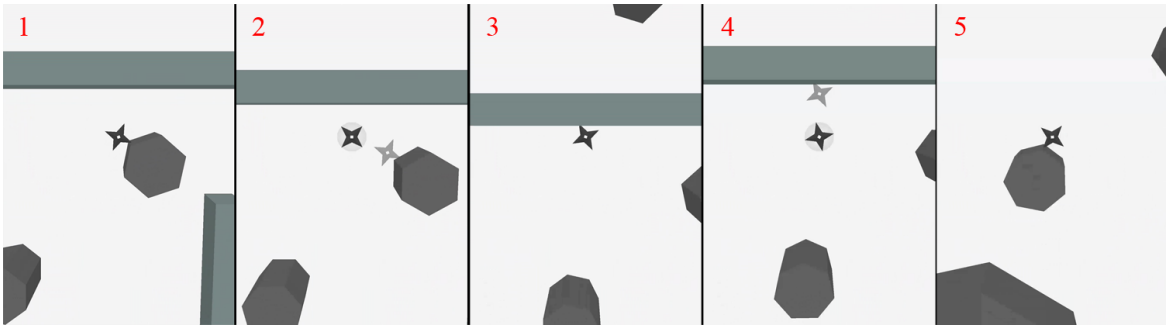

Illustration 24. The player is using a wall to speed up the completion of the level. The simple platform that the shuriken is initially housed by is rotating counterclockwise.

This ability should be and was taken into consideration when designing the puzzles.

4.2.6 The Active Wall

The *active wall* is visually almost identical to the simple wall, the colour being the only difference (Illustration 25).

Illustration 25. A screenshot showing an active wall object housing the shuriken.

The active wall's mechanics are built on top of the mechanics of the simple wall. However, while any other platform type can passively rotate, the active wall can only rotate while housing the shuriken. The rotation is activated by landing on the active wall and deactivated by detaching from it.

4.2.6.1 Usage in Puzzles

The puzzle aspect of the active wall is almost opposite of that of the simple wall. The active wall does not create a barrier that the shuriken is unable to pass because, as a soon as the shuriken lands on the active wall, it rotates and moves the shuriken with it. This allows the shuriken to get to the area which was initially on the other side of the wall. The active wall can be used as a transportation method, which can move the shuriken in a big circle. A similar effect can be achieved via a very big simple platform, which, in fact, has been done in *Shuriken Way*. Nonetheless, the two approaches differ in a number of aspects. Firstly, the detaching mechanics of the active wall and the simple platform are different. The simple platform always sends the shuriken flying away from the platform's center. The active wall, on the other hand, bases the detachment direction on the normal of its side that the shuriken is on. Secondly, the simple platform would constantly rotate compared to the activation-

based rotation of the active wall. Thirdly, the player is able to more freely choose the radius to be rotated at by choosing to land closer or farther from the active wall's center. In contrast, the rotation radius after landing on the simple platform will always be approximately the same because of the platform's shape (Illustration 26).

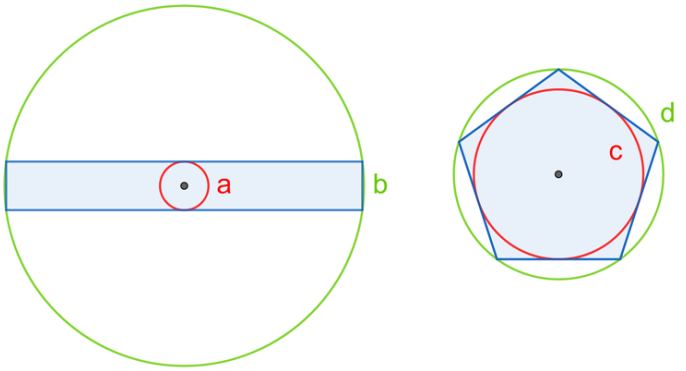

Illustration 26. Circles a and b represent the minimum and maximum rotation radiuses when housed by a wall object. Circles c and d represent minimum and maximum radiuses when housed by a simple platform object. For simplicity, the size of the shuriken is not taken into account here.

This concludes the overview of the platform category of object types in *Shuriken Way*. Nevertheless, there are object types that the player can encounter which cannot house the shuriken. Instead they may try to destroy it. The following subchapter 4.3 describes the enemy objects.

4.3 Enemies

Some game objects seen in the levels of *Shuriken Way* can end the current level in a loss by destroying the shuriken. They are called the enemy objects or simply the enemies. The player is supposed to learn the mechanics of the enemy objects and figure out a way to avoid them. Only 1 type of the enemy objects was implemented for the present thesis, which is the katana.

4.3.1 The Katana

The *katana* object in *Shuriken Way* is inspired by the Japanese swords known under the same name. Real katanas come in different sizes. The game object is modeled after a short

katana sword. The katana model was created using Blender (Illustration 27). Just like the platforms, the katana uses collision detection for its mechanics.

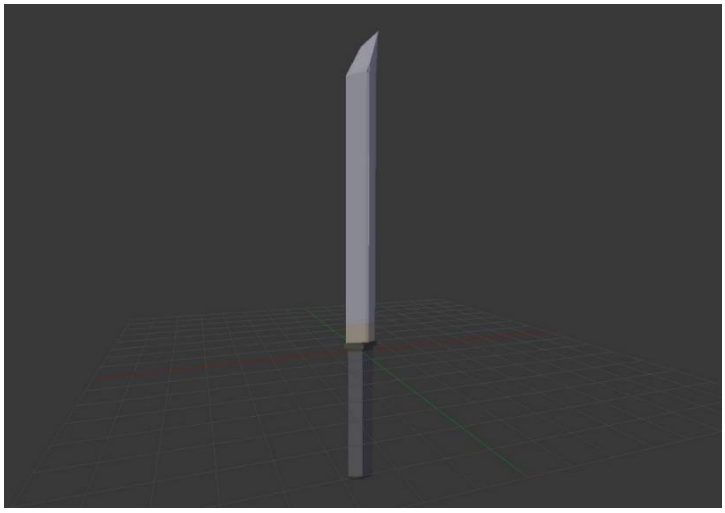

Illustration 27. The 3D model of the katana created using Blender.

4.3.1.1 Cutting

The main mechanic of the katana is cutting the shuriken. It repeatedly swings back and forth through the air waiting for the moment when the shuriken is in the collision area of the katana's blade. The pace at which it swings can be different for each katana. The blade's 2D collision area is only dangerous to the shuriken when the blade itself is in the area. The blade enters and then leaves the area exactly in the middle of the swing (Illustration 28).

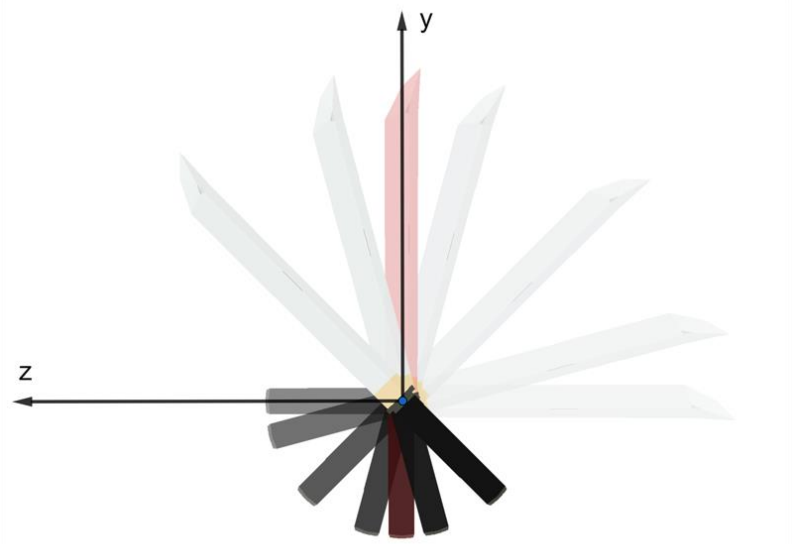

Illustration 28. A side-view example of a katana object's swing. The katana is only dangerous in the middle of the swing, which is marked with red in the illustration. The vectors *y* and *z* represent the *y*- and *z*-axis respectively.

The collision area itself does not move when the sword swings. If the blade of the sword and the shuriken collide, the shuriken is cut into 2 pieces and the level is failed. The collision area of the blade is rectangular and the collision is detected the same way as it is done for the wall type platforms but only using a single collision point placed in the shuriken's center.

4.3.1.2 Bouncing

Collision detection is also used in another mechanic of the katana, which is the bouncing mechanic. If the shuriken enters the bounding area of the katana's handle, the shuriken bounces off. In other words, the velocity of the shuriken is redirected similarly to how bouncing works in the real world. The bounding area of the handle is circular and the collision detection is simply based on the distance between the shuriken's center and the bounding area's center.

4.3.1.3 Usage in Puzzles

The main purpose of using the katana in the levels of *Shuriken Way* is to make them slightly more difficult. It is not that hard to avoid getting hit by a katana but every once in a while the timing is right and the shuriken gets slashed. The katana adds an additional timing-based challenge to the level. Sometimes, when designing a level, a katana object is purposely timed so that one of its swings would happen when the shuriken is expected to be in the collision area. In such situations the player should stay and wait a bit instead of rushing through the collision area of the katana. However, the player is always encouraged to be in a hurry because of the scoring system implemented in *Shuriken Way*. The final score in the level depends on how quickly the player was able to reach all the obligatory collectables. The following subchapter 4.4 provides an overview of the collectable objects in *Shuriken Way*.

4.4 Collectables

Some game objects in *Shuriken Way* are supposed to be collected by the player. These objects are called collectables. Collectable object types can be of 2 varieties: obligatory and optional. The player is required to collect all collectables of the obligatory variety in the level to complete it. However, the player can leave all collectables of the optional variety and still finish the level. Only one type of a collectable object has been implemented during the work of the present thesis, which is the coin.

4.4.1 The Coin

The coin is an obligatory collectable, which means that the player is required to collect all coins in the level in order to complete it. The coin model for *Shuriken Way* was made using Blender (Illustration 29).

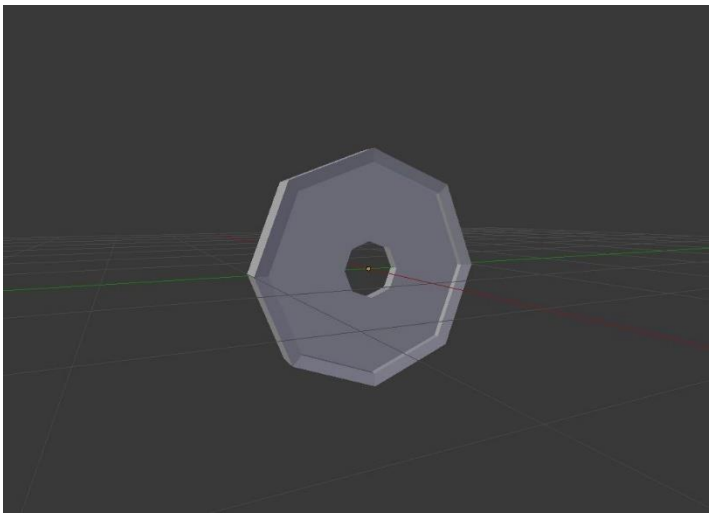

Illustration 29. The 3D model of the coin created using Blender.

Although collecting all coins is required to complete the level, the scoring of how well the player did in the level is still based on the coins collected. This is possible because the value of the coin goes down with time. There are 3 variants of the coin, which are gold, silver and copper, worth 3, 2 and 1 full score points respectively. The time that it takes for the coin to go down in value is level-specific.

4.4.1.1 Usage in Puzzles

The coin object is used in puzzles as a way to visualize one of the destinations that the player needs to reach in the level. Multiple coins can create a path that the player is supposed to follow. Following the coin path the shuriken can encounter objects that are neither platforms or enemies. The next subchapter 4.5 describes this category of objects in more detail.

4.5 Miscellaneous

There are game object types in *Shuriken Way* other than the shuriken that do not belong in the three previously mentioned categories. These can be anything that fits with the rest of the content of a puzzle or the whole game. They can add variety to the levels, which might help increase the time that the player will enjoy *Shuriken Way* before getting bored of it. Only 1 type of an object implemented during the work for the present thesis fits this description, which is the accelerator.

4.5.1 The Accelerator

The accelerator looks similar to an arrow. The model for it was partially created in Blender and a part of it is generated with Android Java programmatically (sphere). Illustration 30 shows what the accelerator looks like in the game.

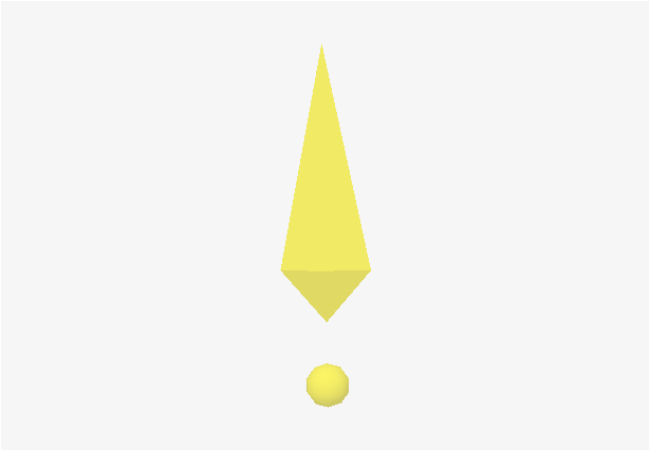

Illustration 30. Full model of the accelerator.

When the shuriken collides with the spherical part of the model, which has a circular bounding area, the shuriken is accelerated in the direction that the rest of the accelerator model (arrow) is pointing to. The collision detection is the same as for the katana bouncing mechanic.

4.5.1.1 Usage in Puzzles

The accelerator object makes it possible for the shuriken to change its direction in mid-air. Although it is perfectly possible to just let the player change the shuriken's direction after landing on a simple platform, it is not exactly the same behaviour. The accelerator does not let the player decide which direction to accelerate to. Another common use of the accelerator in the puzzles is increasing the speed of the shuriken in the air. This often means that with the aid of an accelerator object the shuriken can reach areas or objects in the level which are not possible to reach with just a regular jump.

Multiple game objects arranged in a certain way can form a puzzle. A level in *Shuriken Way* can consist of a single or multiple puzzles. The following chapter 5 briefly describes how the prominent levels of *Shuriken Way* were designed. It also specifies the aspects of them that made them "puzzling" or challenging, as well as how different game objects and mechanics were used to achieve the result.

5 Level Design

Level design is almost synonymous to game design. It is about taking the mechanics and objects implemented in the game and placing them in the level in a pattern which is fun and entertaining to play with. The designer's responsibility is to make sure that there is a good balance between the amount of challenge, the amount of choice and the amount of reward for overcoming the challenge. Getting this right is not an easy task. [7] This will be called level balance further in the present thesis. When designing the levels it is important to understand that challenges which are simple to an experienced player can sometimes be close to impossible for a completely new player. The aim when designing the levels for *Shuriken Way* was to reward the player for overcoming simple challenges (to an experienced player) in the early levels as much as rewarding the player for overcoming difficult challenges later in the game. A total of 9 levels were implemented during the work for the present thesis. These are the levels 1 to 8 and an additional level 12. The following subchapters 5.1 and 5.2 describe the design of a simple short level (5) and a more difficult longer level (12) respectively. The two levels were chosen because they seemed to be more interesting to detail compared to other levels implemented in *Shuriken Way*.

5.1 The Short Level

The chosen short level is the level 5. Orthogonal projection of the level is shown in the Illustration 31.

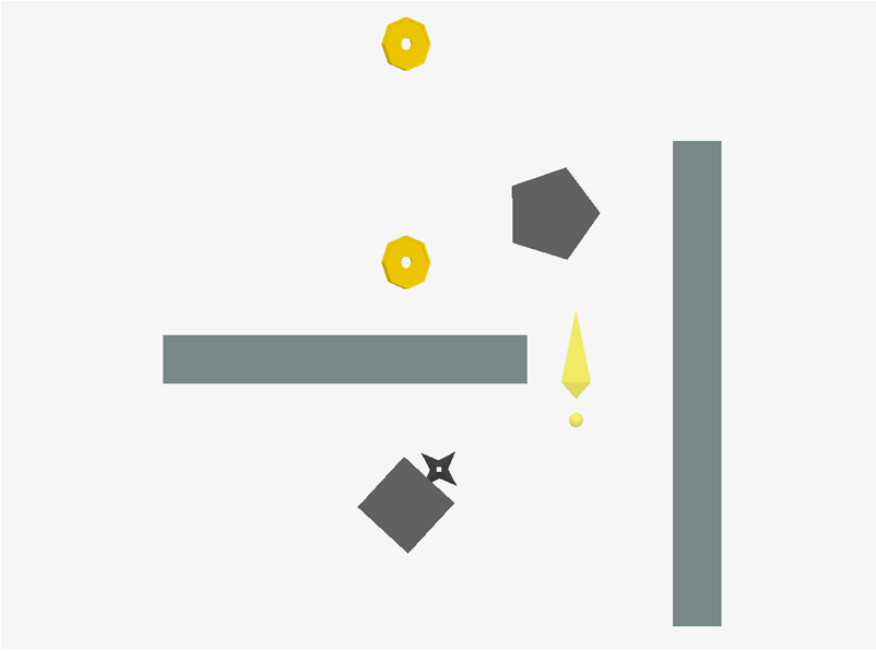

Illustration 31. Orthogonal projection of the level 5 of *Shuriken Way*.

Any terms used further in this subchapter which describe a position or describe an object by its position are referring to the position in the Illustration 31 unless stated otherwise. Any objects mentioned in the subchapter are visible in the illustration.

This level consists of a single puzzle. The first challenge when playing this level is that the player encounters the wall object for the very first time. Collecting the coins without the help of the wall mechanics is likely difficult for a player who has only played the 4 previous levels. This means that the player will most likely have to learn the new mechanics. The obvious first action can be jumping at the accelerator and get accelerated either into the right-most wall or the top-most simple platform. The accelerator put in the level was purposely adjusted so that the next jump after getting accelerated into the right-most wall would most likely put the shuriken in the top-most simple platform. Then it is time for the player to either make use of the wall mechanics or do a precise jump aimed upwards and collect both coins on the way down (Illustration 32 A).

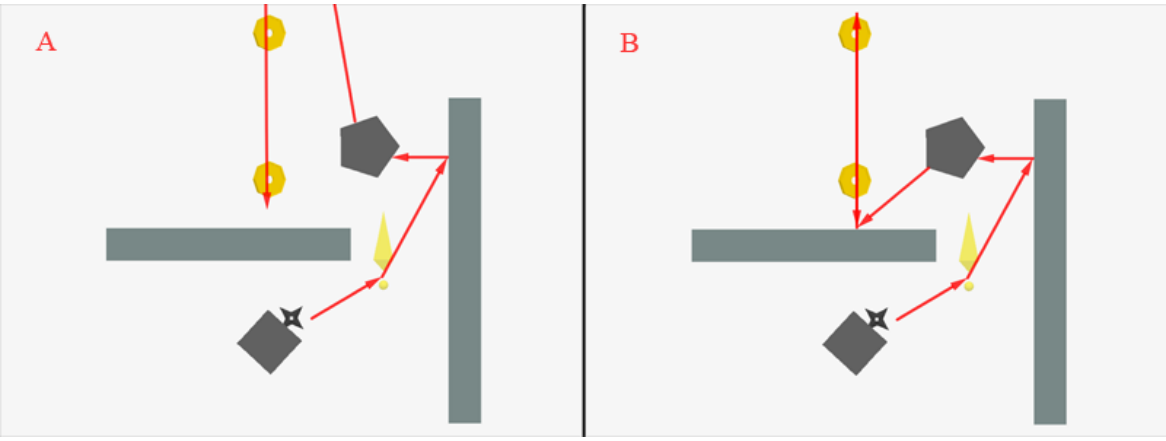

Illustration 32. Intended slow solutions for the level.

Of course, the first option is much easier and all the player is required to do is aim from the simple platform at the left-most wall and land under the bottom-most coin. Another jump up collects both coins and the level is completed (Illustration 32 B). However, this is not going to be quick enough to collect both coins while they are still gold. This is where the optional second challenge of the level comes in. The right-most wall can actually be used to speed up the completion of the level. The lower the shuriken hits the accelerator the lower it is going to stick in the right-most wall. If the player manages to hit a certain area of the wall, it will become possible for the shuriken to skip the simple platform on the next jump and land under the bottom-most coin (Illustration 33).

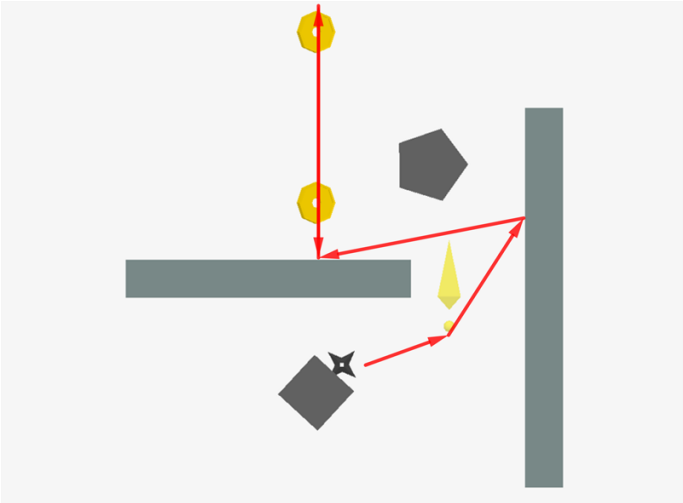

Illustration 33. The fastest intended solution for the level.

If the player is quick enough, then the level will be completed with 2 gold coins collected.

5.2 The Long Level

The chosen long level is the level 12. Orthogonal projection of the level is shown in the Illustration 34. The level consists of 4 puzzles and it is divided according to the Illustration 35.

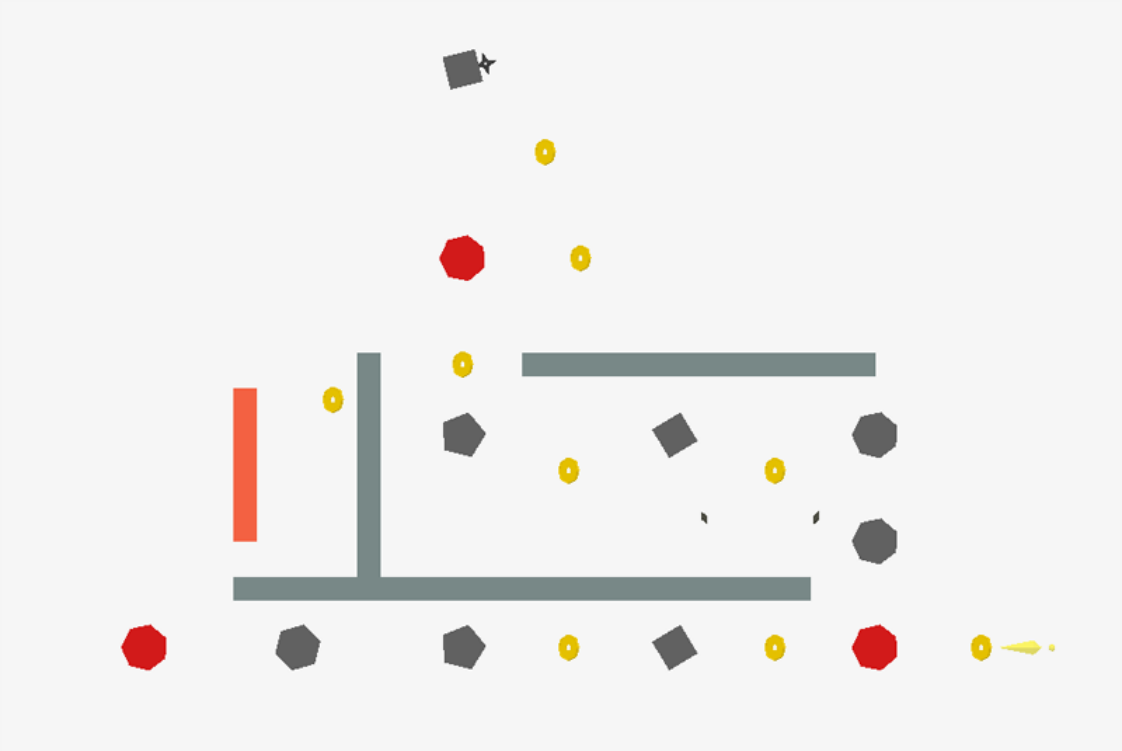

Illustration 34. Orthogonal projection of the level 12 of *Shuriken Way*.

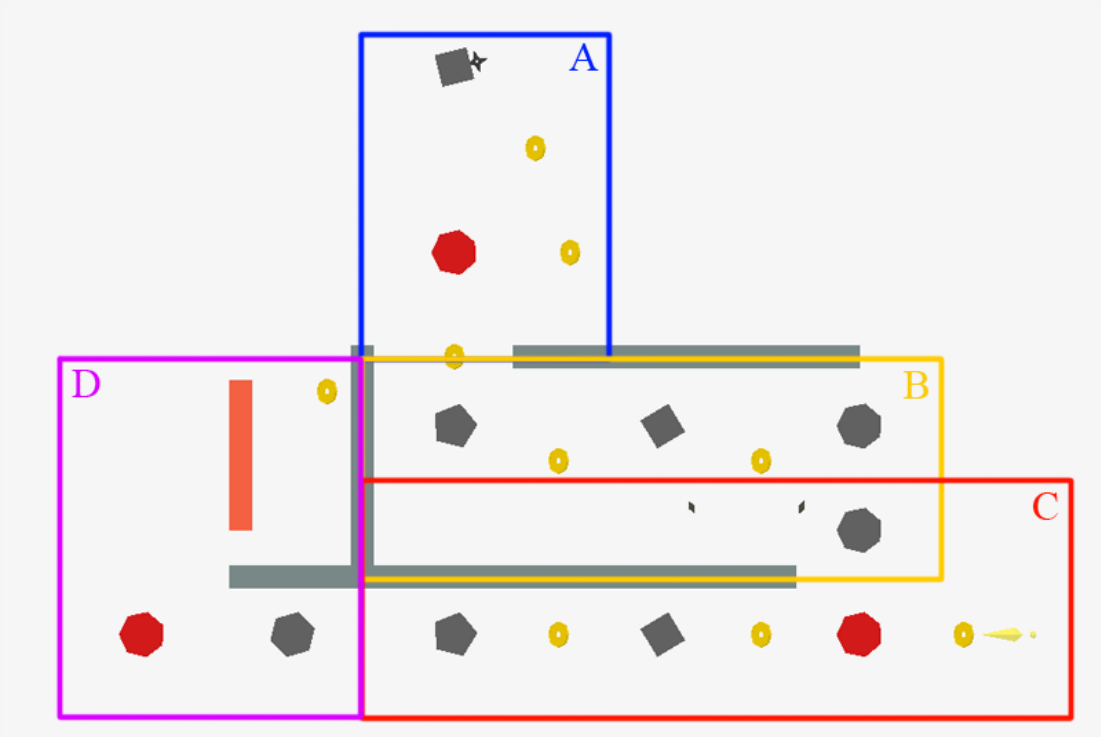

Illustration 35. Division of the level 12 into 4 puzzles where blue (A), yellow (B), red (C) and purple (D) rectangles each encompass areas relevant to a puzzle.

The design of this level makes heavy use of the mechanics of the relativity platform. In comparison to the level 5 described in the previous subchapter, it is now assumed that the player has learned how to use the mechanics of the simple platform and the wall to their advantage. The player has also encountered the active wall and the relativity platform in the previous levels.

Although there is not a definite way to complete the level, an "intended" way that was kept in mind when designing the puzzle does exist. The following subchapters 5.2.1–5.2.4 describe the design of each of the 4 puzzles in the level separately and reveal the "intended" way to complete the level. The term "intended" will be used without quotation marks in these subchapters.

5.2.1 Puzzle A

This subchapter is referring to the Illustration 36 similarly to how the short level was described. Subchapters about the other 3 puzzles are also written the same way, each describing their respective illustration.

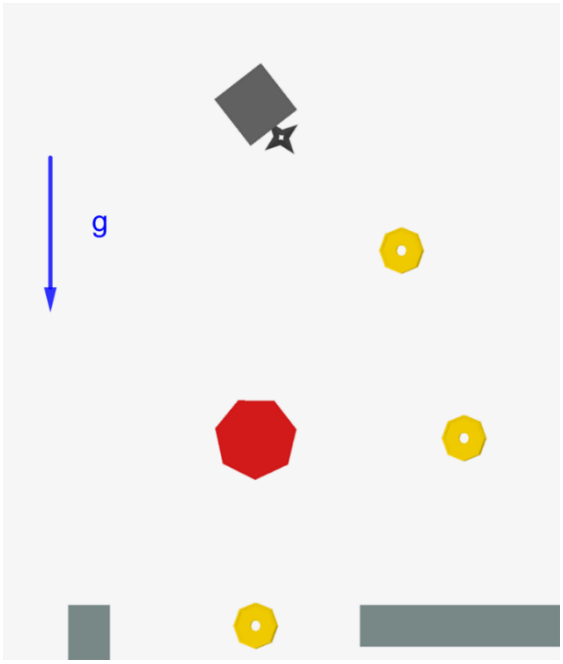

Illustration 36. Area of the level relevant to the puzzle A with the initial direction of gravity for the puzzle marked as *g*.

The main challenge of this puzzle is collecting the 2 top-most coins and then being able to continue with the next puzzle. The most obvious action is to try to hit both coins from the simple platform. This will end with the shuriken getting stuck in the horizontal wall (Illustration 37).

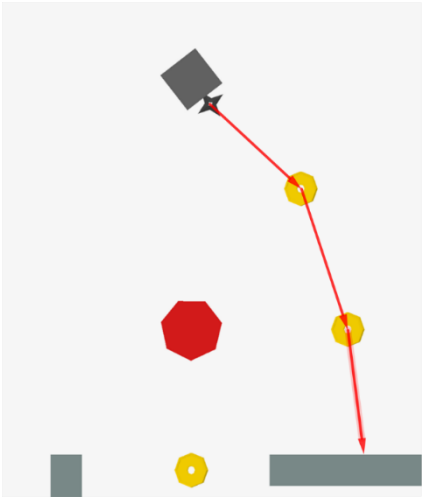

Illustration 37. An incorrect solution of the puzzle.

There is nowhere to go from there. Instead, the player is supposed to make use of the relativity platform. The player should wait for the simple platform to rotate by about 180 degrees and try to throw the shuriken in the direction of the center of the relativity platform. The distance on the horizontal (*x*-)axis between the center of the shuriken and the center of

the relativity platform should be close to 0 for the following trick to work. If the jump (throw) was precise enough, the shuriken should be able to do at least 2 jumps from the relativity platform that will bring it right back to the same relativity platform. This is because the relativity platform rotates both the shuriken and the gravity around the same axis and at the same speed. This means that the direction of the jumps will be approximately opposite to the direction of gravity. Collecting all 3 coins in the puzzle and continuing with the puzzle B is not difficult from there (Illustration 38).

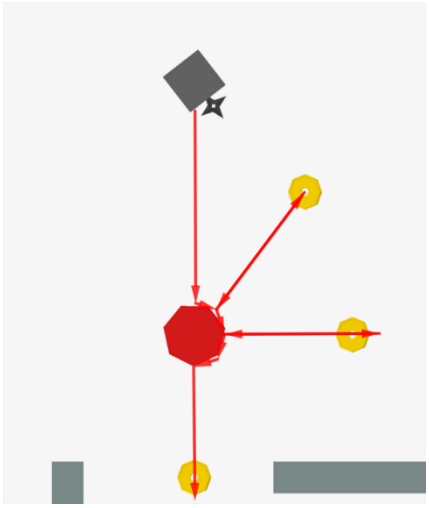

Illustration 38. The intended solution for the puzzle A.

After the bottom-most coin is collected the player moves on to the puzzle B.

5.2.2 Puzzle B

The area of the level relevant to the puzzle B is in the Illustration 39. The partially visible coin at the top of the illustration is intended to be collected during the puzzle A so it can be ignored.

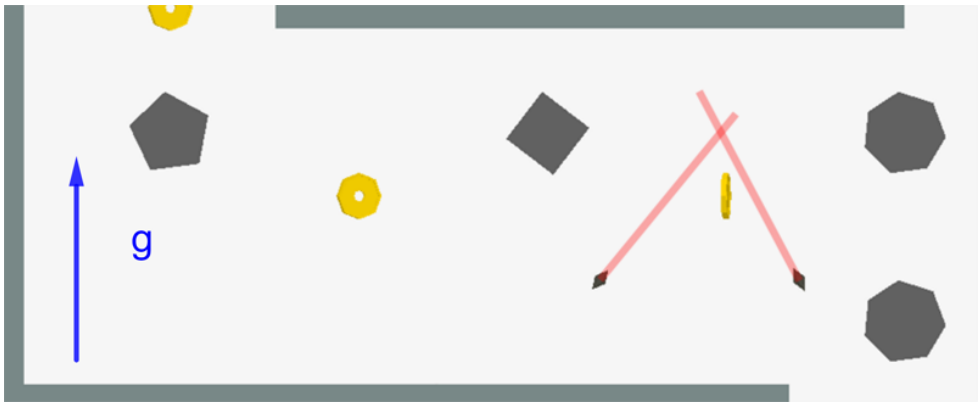

Illustration 39. Area of the level relevant to the puzzle B with the initial direction of gravity for the puzzle marked as *g*.

After completing the puzzle A using the intended solution, the gravity vector should point approximately upwards in the illustration. The shuriken should be housed by the left-most simple platform. Otherwise it is still possible to complete the puzzle B but it might not be as intuitive as the intended solution described further. The only obligatory challenge for the player to overcome in this puzzle is avoiding the katana objects, which look diamond-shaped in the illustration. The partly transparent red areas that intersect with the katanas represent the collision areas of their blades. The shuriken can get hit by the katanas when collecting the right-most coin. Other than that, completing this puzzle is assumed to not be difficult because it is very similar to what the player encountered in the previous levels of *Shuriken Way*. There are many different ways for the player to complete the puzzle. The left-most simple platform was purposely put rotating counterclockwise. The most obvious action from the player is to wait for the platform to fully rotate and throw the shuriken in the direction of the left-most coin (fully visible in the illustration) to collect it. An optional challenge for the player would be to save some time. Instead of waiting for the platform to fully rotate, the shuriken can jump at a specific part of the vertical wall, jump off the wall, hit the coin and land above the simple platform second from left (Illustration 40).

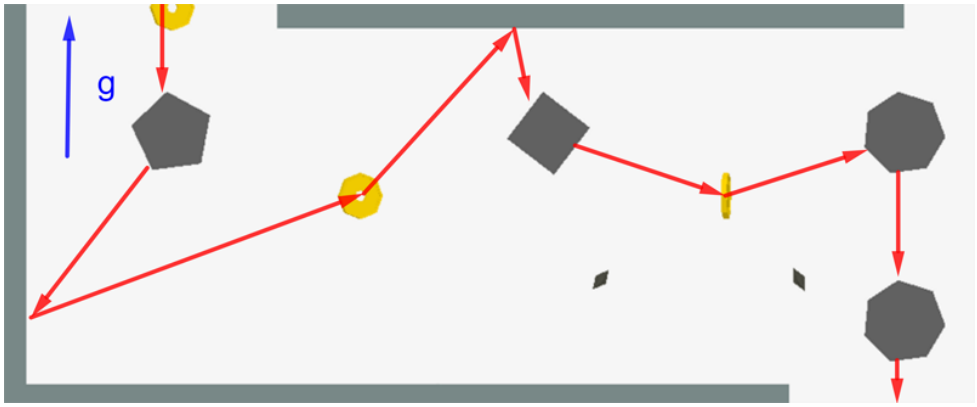

Illustration 40. The fastest intended solution for the puzzle B.

Slightly adjusting the direction of gravity in the first puzzle might even help land directly on the simple platform after collecting the coin. To complete the puzzle the shuriken should get past the katanas while collecting the right-most coin. To continue with the puzzle C the player should move the shuriken to the bottom-most simple platform.

5.2.3 Puzzle C

The area of the level relevant to the puzzle C is in the Illustration 41. The puzzle C starts where the puzzle B ends. In the current illustration it is the top-most simple platform that the shuriken is housed by.

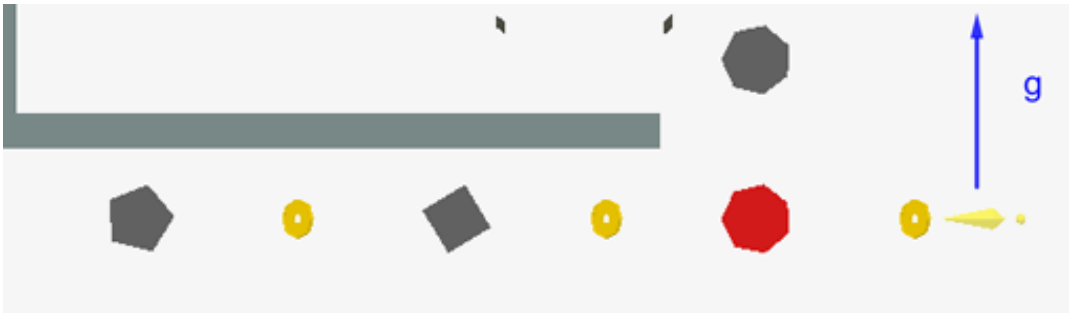

Illustration 41. Area of the level relevant to the puzzle C with the initial direction of gravity for the puzzle marked as *g*.

The 2 katana objects visible on the illustration are from the previous puzzle and can be ignored. If the intended way to complete the level was followed until this point, then the direction of gravity should still be approximately upwards in the current illustration. In this puzzle the player encounters another relativity platform. The challenge here is similar to that of the puzzle A. How the shuriken will land on the relativity platform is going to affect the difficulty of collecting the right-most coin. The intended solution here starts with landing as directly on top of the relativity platform as possible. The relativity platform is rotating clockwise. After the shuriken lands on the relativity platform the gravity, which is currently upwards, will start rotating clockwise. The direction of gravity when jumping to collect the right-most coin will be approximately to the right in the illustration. This means that the shuriken will jump approximately in the direction of gravity. The accelerator to the right of the relativity platform is supposed to get the shuriken back onto the relativity platform. The player should then wait for the direction of gravity to rotate until it is directed approximately to the left in the illustration and collect the rest of the coins in the level. The full intended path is shown in the Illustration 42.

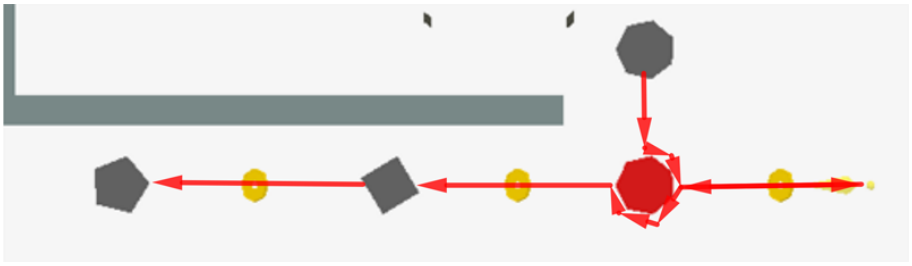

Illustration 42. The intended solution for the puzzle C.

Jumping from the left-most simple platform to the one to the left of it, which is not visible in the current illustration, will complete the puzzle and move to the final puzzle D.

5.2.4 Puzzle D

The area of the level relevant to the puzzle D is in the Illustration 43. If the intended solution was followed until this point, then the direction of gravity should be approximately to the left in the illustration. The shuriken should be housed by the simple platform.

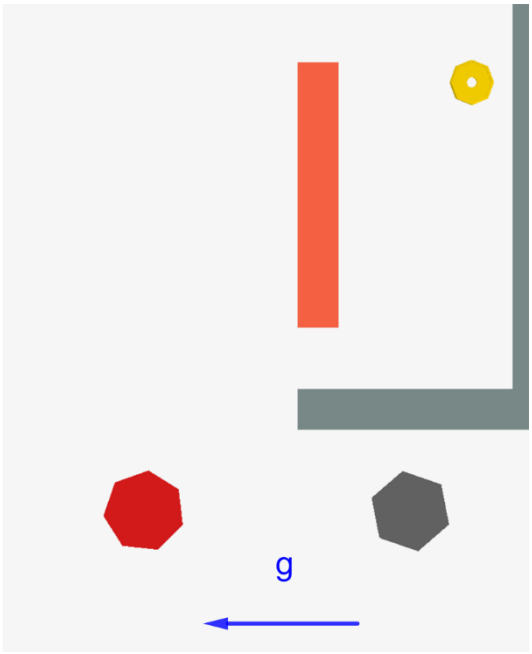

Illustration 43. Area of the level relevant to the puzzle D with the initial direction of gravity for the puzzle marked as *g*.

The challenge here is to understand that the reachability of the active wall from the relativity platform depends on the direction of gravity during the jump. Jumping in the opposite direction to gravity will not get the shuriken very far in the wanted direction. This is because the shuriken will be pulled in the direction opposite to the jump. For the reason of this being the end of a long level, the jump in the opposite direction to gravity can still successfully reach some of the active wall. Otherwise the level would become more difficult than intended. This solution is shown in the A section of Illustration 44.

However, the recommended solution here would be landing the jump from the simple platform to the relativity platform slightly lower than the center of the relativity platform. Another jump should be made when the direction of gravity points approximately downwards in the illustration. The shuriken should then reach the active wall. The player has encountered the active wall objects in the previous levels and is assumed to have no trouble collecting the last coin in the level. The intended solution for the level was tested to

be quick enough to collect all coins while they are gold. The path of the solution is shown in the B section of the Illustration 44.

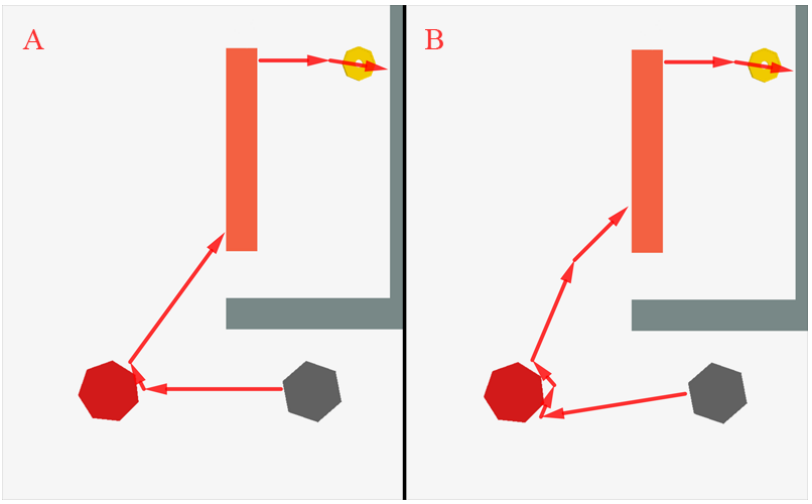

Illustration 44. The intended solutions of the puzzle D.

Level design is a very important aspect of the game but it can lose all meaning if the player is not able to fully experience it because of performance or compatibility issues or bugs. The player might also dislike or not understand the design. This is why it was important to test *Shuriken Way* before concluding the work for the present thesis. The following chapter 6 focuses on the testing stage of the work.

6 Testing

As with developing any type of software, assuring high quality of the developed video game is very important. This requires testing both the performance and the functionality throughout the development. Moreover, when developing a video game, it is very easy for the developers to leave very important issues completely unnoticed or ignored when giving too much attention to improving other aspects of the game. Although they may seem very important to the developers and might be more interesting to solve, the future players of the game are the ones to judge what is important and what is not. The developers can imagine that the player experience will be great but it is very important to test if the fantasy actually represents reality. Very often it does not and testing the game helps the developers target critical issues present in the game. [7] Three types of tests have been conducted during the work for the present thesis. These were compatibility tests, performance comparison with alternatives and lastly playtests with new players. The following subchapters 6.1–6.3 provide the description of the tests, the results and the analysis of the results. Some of the mentioned subchapters also list the improvements implemented or to be implemented based on the results of the conducted tests.

6.1 Compatibility

During the development of *Shuriken Way* the game was frequently tested on an emulator of LG Nexus 5X (2015) with Android 7.0 installed and an actual LG Nexus 5 (2013) with Android 6.0.1 installed. The definition of the term emulator is in the Glossary (Appendix I). In addition, the game was compatibility-tested on more devices when the game was in a playable state, meaning that the following was true:

- Several levels could be started, played and completed.
- Mechanics of the game objects in the levels were implemented.
- The 3D models of the game objects were being rendered.
- The game was performing well on the LG Nexus 5 used throughout the development.

Another set of compatibility tests was conducted after implementing fixes for some of the issues found during the first set and playtesting. The following 6 devices were used for both compatibility test sets:

1. Huawei MediaPad T3 10 (2017) with Android 7.0
2. Xiaomi Redmi Note 4 (2017) with Android 7.1.2

3. Samsung Galaxy Grand Prime (2014) with Android 5.0.2
4. OnePlus 3 (2016) with Android 8.0.0
5. OnePlus 5T (2017) with Android 8.0.0
6. LG G4c (2015) with Android 6.0

To measure the performance on these devices a modified version of the game was used, which collected data for the first 5 minutes of playing the game and then output it on the screen (Illustration 45).

Illustration 45. A screenshot from the modified version of the game taken on the OnePlus 3.

A normal version of the game was also tested on these devices to discover issues that were not related to performance.

6.1.1 The Results

The initial set of conducted compatibility tests uncovered 1 critical bug and no performance or visual issues. The found critical bug was that it was not possible to start the game from landscape mode on the Huawei MediaPad T3 10 because doing so caused the game to exit. With the LG Nexus devices used throughout the development, starting the game while in landscape was not something that was encountered. Unlike Huawei MediaPad T3 10, these devices do not allow switching the app launcher (see Appendix I for definition) to landscape mode. However, switching to landscape mode in Chrome web-browser application and then starting *Shuriken Way* from the multitasking menu showed that the issue is also present on

the LG Nexus 5. As for performance of the game on these devices, the results of the performance tests are in the following Table 1.

Table 1. The results of the performance tests on different devices.

Device	**Screen resolution**	**Average frametime (ms)**	**Average update time (ms)**	**Noticeable visual stuttering**
LG Nexus 5	1080 × 1920	3.9	0.08	No
Huawei MediaPad T3 10	800 × 1280	3.5	0.11	No
Xiaomi Redmi Note 4	1080 × 1920	15.1	0.1	No
Samsung Galaxy Grand Prime	540 × 960	3.4	0.07	No
OnePlus 3	1080 × 1920	13.9	0.27	No
OnePlus 5T	1080 × 2160	14.3	0.16	No
LG G4c	720 × 1280	6.7	0.55	No

Frametime is the time it takes for the game to calculate a frame. *Update time* is the time it takes the game to update the game state. *Shuriken Way*, like most Android games, caps FPS (frames per second) at 60, which requires frametime less or equal to $1000 / 60 \approx 16.67$ milliseconds. Average frametime measurements in the third column of the table show that all devices on average successfully run the game at 60 FPS. The game state is updated 40 times per second and the average update times in the fourth column of the table show that on average all tested devices successfully kept up with that. What is very interesting about the results is that OnePlus 5T, which has the best performing GPU (Adreno 540[18]) according to online benchmarks[19], had one of the highest average frametimes. This might be because newer devices or versions of the Android operating system might be better at saving resources when working at full power is unnecessary. However, average frametime is not everything that the performance of a video game can be judged by. There can be short periods during which the game is performing bad. These are called *visual stuttering* in the

[18] https://www.gsmarena.com/oneplus_5t-8912.php

[19] https://www.notebookcheck.net/Smartphone-Graphics-Cards-Benchmark-List.149363.0.html

present thesis. It can be argued that visual stuttering is even a bigger issue than low FPS to the player. None of the tested devices had noticeable issues with visual stuttering, which was determined by playtesting the game and the observation results can be seen in the last column of the Table 1. The second set of compatibility tests conducted after implementing the fixes for some of the issues found during the first set and playtesting with new players confirmed that *Shuriken Way* was still very compatible with all 6 testing devices (Appendix VII).

6.1.2 Implemented Improvements

The critical bug of not being able to start the game from landscape device orientation was fixed. The issue was caused by the the fact that the Android operating system initially tries to start the application in the orientation that the system is in, in this case it is landscape. The application can then force a specific orientation. When switching to the forced orientation the method *onDestroy* specified by the application is called. The method *onDestroy* is also called by the default functionality of the back button, which is why it was overridden in *Shuriken Way*. The *onDestroy* method specified by *Shuriken Way* was forcing the application to completely exit, which was necessary because of the following reason: The data stored on the GPU was deleted by the default functionality of the back button and not restored on the next start of the application. However, the application itself was started in the same state the method *onDestroy* was called in. This meant that the application still assumed that all the data stored on the GPU was available. Forcing the application to fully exit in the *onDestroy* method meant that the application would start next time without restoring its previous state and reload everything like it is supposed to. But since it turned out that the method *onDestroy* is also called when switching to the correct orientation, the application was also forced to exit when it was started from landscape. To fix this the application would no longer be forced to exit in the method *onDestroy*. Instead, it would be done directly in the overridden functionality of the back button.

6.1.3 Future Improvements

The compatibility issue found during the tests was fixed so there is not anything specific planned to be improved in the future beyond the scope of the work for the present thesis. However, the game is very likely going to be tested on more devices as the development progresses and the new issues discovered are going to be fixed.

6.2 Alternatives

Comparing the performance of *Shuriken Way* against other similar Android games was different from measuring performance of the game on different devices. Modifying the source code to measure performance of *Shuriken Way* for compatibility tests was not difficult. However, doing the same for the similar games was not a reasonable choice. At the time of the testing there were Android applications for measuring the framerate of games but all found applications were either not doing what they claimed to do, not suited for collecting data or not free (see Appendix VIII). A free alternative method was found instead. With the use of GAPID (Graphics API Debugger)[20] the performance of all 3 games, including *Shuriken Way*, was measured and the results were compared.

6.2.1 GAPID

Like the name suggests, Graphics API Debugger is a collection of tools that can be used for debugging an application's use of the graphics API. GAPID can debug any application that is configured in its manifest as debuggable. GAPID was not made specifically for measuring graphics performance of applications but rather for tracing their work. However, GAPID allows setting the number of frames to render after which it should stop tracing. When the tracing is finished, it outputs the time it took to trace the set number of frames. The approximate average frametime for the trace can be calculated from that info by dividing the time it took to render and trace the frames with the frame amount. GAPID slows the graphics down quite a bit so it does not measure the real performance of the graphics. This is why using this method is definitely not a perfect solution. Comparing different games using this method was still possible though. The only requirement was setting the "debuggable" option to "true" in the application manifests of the games. The manifests of the alternative applications were modified with the help of the Apktool[21].

6.2.2 The Results

A total of 27 GAPID tests have been conducted – 9 for each game. The tests were run on the LG Nexus 5. The summary of the results is presented in the Table 2. The full table of results is available in the Appendix IX.

[20] https://github.com/google/gapid

[21] https://ibotpeaches.github.io/Apktool/

Table 2. Summary of the results of the conducted performance comparison tests.

# of tests	**# of frames**	**Average time (s) for**		
		Shuriken Way	***Ninja Star!***	***Shuriken***
3	5000	116.00	117.11	107.96
6	10000	252.81	239.61	258.75
Total average frametime (ms):		**24.87**	**23.85**	**25.02**

The average frametimes in the last row of the table show that *Shuriken Way* is on average a better graphical performer than *Shuriken* but a worse performer compared to *Ninja Star!*. Both *Shuriken* and *Ninja Star!* are simple 2D games, so good frametime is expected from them compared to a game with 3D graphics like *Shuriken Way*. However, a game with a lot of visual stuttering can still have a very good average frametime. The graphical performance might still be excellent between the periods of stuttering. These can be caused by anything in the program, not just graphics. This means that GAPID does not necessarily make the stuttering longer but does slow down the graphics, so the effect of the visual stuttering on the average frametime might become imperceptible. Playtesting all 3 games without GAPID showed that *Ninja Star!* had frequent and noticeable visual stuttering while *Shuriken Way* and *Shuriken* did not. Taking everything mentioned into account, *Shuriken Way* did not lose to the alternatives in terms of performance.

6.3 Playtests

The most important type of testing to game designers when developing a video game is letting people playtest the game. New players see the game with “fresh eyes” and can point out issues in the game that the designers and the developers have gotten used to while working on the game. It is also important to let the players test the game multiple times to make sure that it can be enjoyed more than once. In some cases it gives the players time to reach the later stages of the game and provide useful feedback. Otherwise a game with strong initial appeal can still be created but the appeal might have a short lifespan if the game has not been properly tested. [7]

During the work for the present thesis the testers were allowed to play *Shuriken Way* for as long as they found was necessary for them to provide feedback. A total of 6 players have

tested *Shuriken Way*. All of the testers had previously played mobile games. The testers were asked via a questionnaire (Appendix X) to give feedback on the overall gameplay, performance, level balance, controls, mechanics and bugs. The testers were also asked in the same questionnaire for any kind of suggestions for the further development of *Shuriken Way*. The following subchapter 6.3.1 details the results of the conducted playtests. The further subchapters 6.3.2 and 6.3.3 describe the improvements made or to be made based on the results.

6.3.1 The Results

The conducted playtests proved very useful. A lot of issues were reported and many suggestions were made. Overall the players seemed to feel positive towards *Shuriken Way*. The testers were expecting issues in the game because they knew the game was still in development. However, real players will expect a game with little to no issues, so all feedback collected during the playtests, positive or negative, should be considered and used for future improvement. The following subchapters list the results of the conducted playtests. Raw response data from the questionnaire (*questionnaire_data.csv*) is available in the Accompanying Files (Appendix IV).

6.3.1.1 Overall Gameplay

In terms of the overall gameplay not much was criticized that was not expected. The players did not like that restarting the level required using the navigational back button. Adding a separate restart button on the screen was already planned for the future development. However, using the back button on the LG Nexus 5 used throughout the development was not too bothersome. This is because the LG Nexus 5 has an on-screen back button instead of a physical one. Physical buttons are more awkward to press for most Android users according to a poll run by Android Authority in 2017 [11]. The playtests helped understand how much more important than initially thought the separate restart button really is. The same goes for the separate menu button. The players rated the overall gameplay on a scale of 1 to 5, where 1 and 5 represent “awful” and “perfect” respectively. The results of the rating are plotted in the Illustration 46.

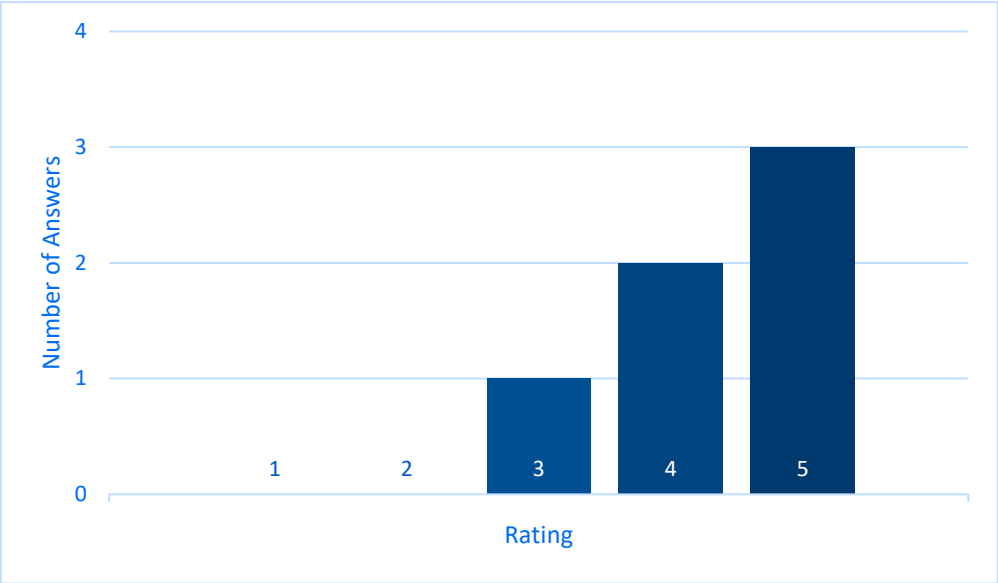

Illustration 46. The results of the overall gameplay rating on a scale of 1 to 5.

This makes the average rating about 4.3 out of 5, which means that there is definitely room for improvement.

6.3.1.2 Performance

Most players were happy with how *Shuriken Way* performed. No specific performance issues were pointed out. The players rated the performance on a scale of 1 to 5, where 1 and 5 represent “awful” and “perfect” respectively. The results of the rating are plotted in the Illustration 47.

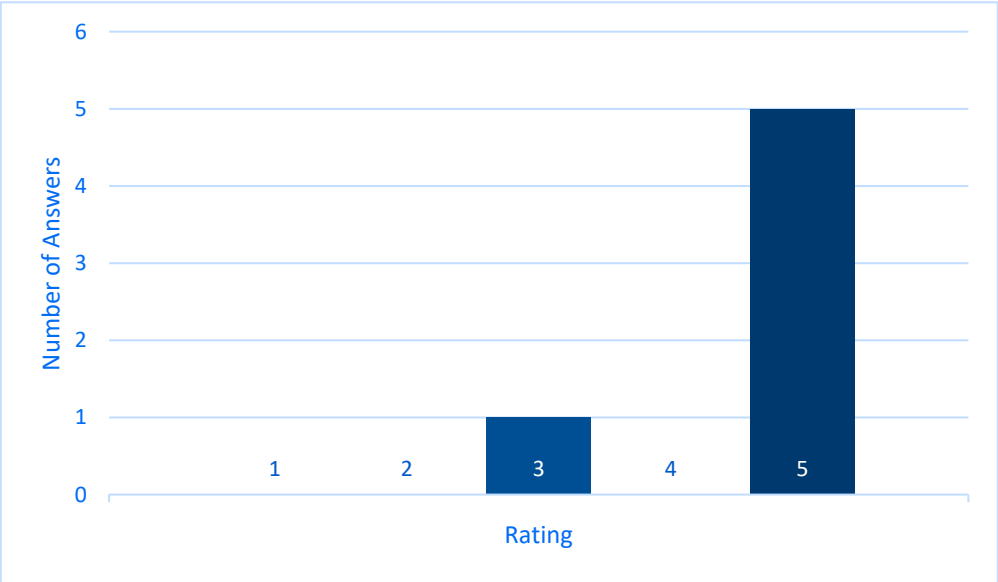

Illustration 47. The results of the performance rating on a scale of 1 to 5.

This makes the average rating about 4.7 out of 5. Only one tester gave a rating for the performance which is not 5. The tester did not specify any reason for doing so.

6.3.1.3 Controls

Most of the testers said that the controls were confusing at start. However, the confusion cleared up after trying to approach the game in different ways. This can still make the game less appealing to new players and the issue should be addressed.

6.3.1.4 Confusion

There were also the following aspects of *Shuriken Way* that the players found confusing:

- It was not obvious what was needed to be done to finish a level.
- The lengths of the levels were unclear.
- The direction of the jump of the shuriken was not intuitive enough for the testers to fully figure it out on their own.

Other than that everything seemed to be understandable for the testers. This built a clearer picture of what the game should be helping the new players with.

6.3.1.5 Level Balance

Most of the testers thought that the level balance needed improvement. They rated the level balance on a scale of 1 to 5, where 1 and 5 represent “awful” and “perfect” respectively (Illustration 48).

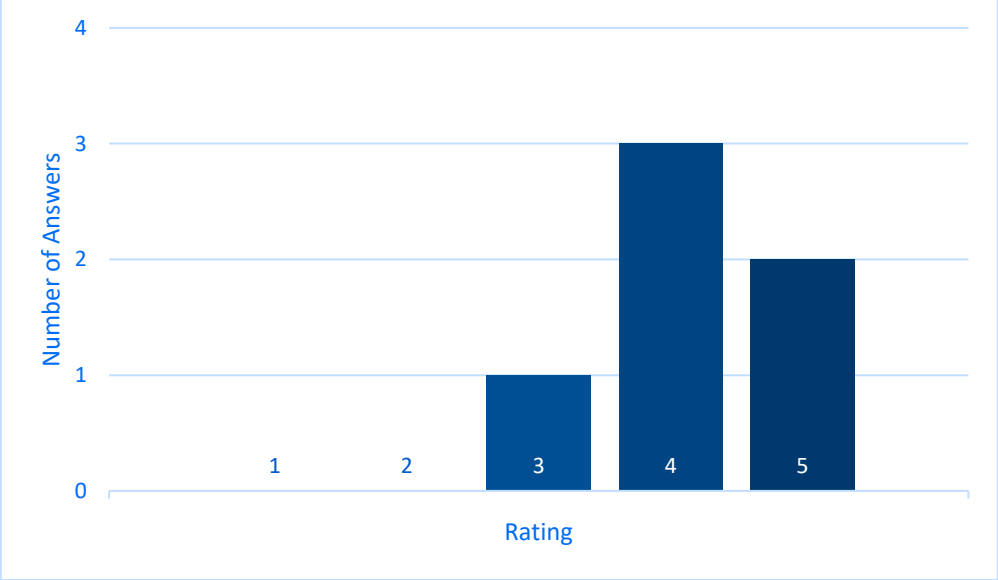

Illustration 48. The results of the level balance rating on a scale of 1 to 5.

This makes the average rating about 4.2 out of 5. The following suggestions for improving the level balance were made by the testers:

- The levels should be shorter in the beginning and new mechanics should have their own dedicated levels that introduce them.
- A new game object type should not appear at the end of a long level.
- There should be tips in the levels where the player encounters new mechanics for the first time.

All of the suggestions were taken into strong consideration.

6.3.1.6 Game Mechanics

Most of the testers liked most of the game mechanics but half of the testers disliked at least one of the mechanics. The mechanic that the testers seemed to dislike the most was the gravity and camera rotation caused by the relativity platform. The mechanic might be too confusing to a new player. The only level implemented for the present thesis that contained the relativity platform was level 12. It was added in order to demonstrate how a difficult level later in the game might look like. The levels 9, 10 and 11, which were not implemented for the present thesis, would give the players a better introduction to the relativity platform. The camera rotation made one of the testers feel nauseous. It is reasonable to assume that it can happen to more players in the future. Another mechanic that one of the testers pointed out was the collision detection of the accelerator. It was unclear that in order to get accelerated the shuriken needs to intersect with the spherical part of the accelerator and the rest of the model is only there to show the direction of the acceleration. This made enjoying the game object difficult for the tester.

6.3.1.7 Bugs

The following bugs were reported by the testers:

- The buttons in the menu screen did not always register the touches.
- Going through walls was possible under some conditions.
- Collision detection and detaching did not work properly on the corners and the shorter sides of the walls.
- The level 12 did not always end in a loss when the shuriken fell into the void.

The low number of bugs found during the playtesting might mean that either the game does not have that many bugs or that there was not enough testing done. No actual compatibility issues were reported by the testers.

6.3.1.8 Suggestions

The following suggestions were made by the testers:

A. The levels should restart automatically in some cases.
B. Longer levels should have save points.
C. The rotating coins on the interface should be removed because they were apparently useless or distracting.
D. There should be separate restart and menu buttons on the screen.
E. There should be a button to continue with the next level.
F. There should be tips or tutorials explaining new mechanics.
G. There should be helping elements rendered in the early levels showing how the jumping directions work on different platforms.
H. There should be the title of the game rendered in the menu screen even if the interfaces are far from finished.
I. Level 12 should be less intense.
J. There should be sounds in the game.

Although most of the suggestions were obvious and already planned for the future development of *Shuriken Way* (D–H, J), some of them were very unexpected and were definitely taken into consideration (A–C, I).

6.3.2 Implemented Improvements

All reported bugs have been fixed. The first bug that was fixed was the ability to go through the walls under certain conditions. The second one was detaching from the shorter sides or the corners of the walls not working the way it was intended. Both of these bugs were caused by the same logic error. The part of collision detection which determines whether the bounding circles of the shuriken and the wall intersect was not programmed correctly. The bounding circle for the wall was just based on the longest of the wall sides. The bounding circle of the shuriken was effectively of radius 0. This is why, when trying to land on a shorter side, the shuriken only attached to the wall when the center of the shuriken was already inside the wall. Being inside the wall is not expected from the shuriken and the algorithm that determines the side of the wall that the shuriken is on is not suited for it. In this case the algorithm always determines the side as right because it is the default output (see subchapter 4.2.5.2). Obviously, the detaching mechanic was also affected by this.

Basing the bounding circle of the wall on both width and height of the wall and using the correct bounding circle for the shuriken fixed both of the issues (Illustration 49).

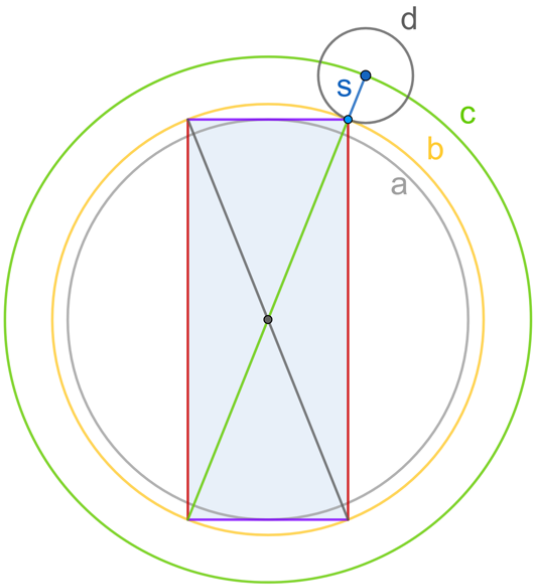

Illustration 49. The rectangle represents a wall object. Circle *a* is the bounding circle that would be used for the wall before the fix. Circle *b* is the correct bounding circle of the wall. Circle *d* is the bounding circle of the shuriken with radius *s*. Circle *c* represents the maximum distance that the center of the shuriken can be at for the bounding circles to intersect.

The third bug that was fixed was that the level 12 did not always end when the shuriken fell into the void. It was caused by the relative platform objects rotating the gravity vector. The mechanic of falling into the void was developed before the relativity platform was designed. The program was comparing the shuriken's negative coordinates on the vertical (*y*-)axis to the level-specific maximum. To fix the issue the coordinate of the shuriken in the direction of gravity would be used instead. Finally, two more changes were made to fix the last bug found during the playtests. The bug was that the buttons in the menu screen did not always register the touches. Firstly, the distance that the player could drag a finger vertically on the screen until it was considered scrolling was slightly increased. Secondly, horizontal dragging would no longer cancel the touch at all.

Besides bugs, some other changes, which were comparatively simple, were also implemented. The issue that the coin counters were distracting to one of the testers was addressed. The size of the coin counters and the speed at which the coin models were rotating were decreased, potentially making them less distracting. The title of the game was added to the menu screen.

While fixing the collision detection for the wall, some additional optimisation was implemented. A new bounding rectangle around the wall was added with width, height and rotation being equal to those of the wall but with the diameter of the shuriken's bounding circle added to the width and the height (Illustration 50).

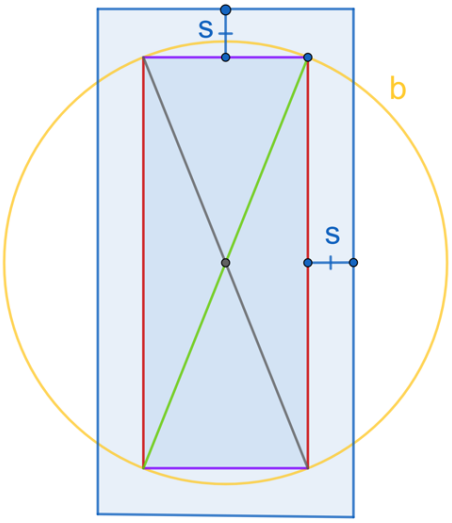

Illustration 50. The smaller rectangle represents a wall object with the bounding circle *b*. The bigger rectangle represents the new bounding rectangle. Length *s* is the radius of the bounding circle of the shuriken.

After passing the bounding circle intersection test, the shuriken's center would then also be checked for intersection with the bounding rectangle. The bounding circle of a wall is usually quite big compared to the wall, which means that passing the bounding circle intersection test while not actually intersecting with the wall is very common.

6.3.3 Future Improvements

Most of the other issues and suggestions collected during the playtesting with new players will be fixed or implemented during the future development of *Shuriken Way*, which is beyond the scope of the work for the present thesis. This includes but is not limited to the following:

- better interfaces;
- separate restart, next level and menu buttons;
- reworking the accelerator object;
- adding game sounds;
- improving the level balance;
- adding tips and helping elements for new mechanics;

- showing the whole level before the game starts;
- designing more game objects;
- designing more levels;
- conducting more playtests with more players.

It is also certain that the relativity platform will be used a lot less than initially planned when designing new levels in the future.

7 Conclusion

During the work for the present thesis a playable version of a video game for Android was developed. The name of the game is *Shuriken Way*. The game was developed without the use of game engines. Several game objects with associated game mechanics were designed and implemented. The design of the objects allowed them to be combined into patterns to form physics-based puzzles. The game objects were then used in the design of 9 game levels which were implemented in *Shuriken Way*. Each implemented level contains a single or multiple puzzles. Android games similar to *Shuriken Way* were found and compared to *Shuriken Way* to make sure that the developed game could provide a unique player experience.

Several types of tests were conducted to discover bugs, compatibility and performance issues. This included comparing the performance of *Shuriken Way* to the found similar games. Completely new players, who had previously played mobile games, were asked to playtest the game and provide feedback via a questionnaire. The testers provided their opinions about the gameplay, performance, game mechanics, controls, clarity and level balance. Overall the testers seemed to like the game but some aspects of it were criticized. The testers also reported bugs and gave suggestions for the future development of *Shuriken Way*. All of the reported bugs were fixed. Some of the other types of issues that the testers have pointed out were also fixed. Finally, another set of compatibility tests was conducted to make sure that the changes made did not cause incompatibility with the devices that *Shuriken Way* had previously been tested on. The other suggestions made by the testers and unsolved issues were considered when planning the future development of *Shuriken Way* beyond the scope of the work for the present thesis.

The choice of not using game engines was justified. The game is compatible with devices of different kind (smartphones and tablets) and from different manufacturers. The performance is close to perfect on all devices that the game was tested on.

Special thanks go to the testers for taking part in the testing stage of the work and to the supervisor Raimond-Hendrik Tunnel for helping drastically improve the quality of the thesis by providing useful suggestions and feedback. Thanks also go to the people behind Computer Graphics (MTAT.03.015) course of the University of Tartu, who did a great job teaching low-level techniques for rendering computer graphics. The knowledge acquired during the course was extremely helpful when doing the work for the present thesis.

8 References

[1] Viennot N., Garcia E., Nieh J. A Measurement Study of Google Play. ACM Digital Library. 2014, pages 8–10. https://doi.org/10.1145/2637364.2592003 (21.11.2017)

[2] Zioma R., Pranckevičius A. Unity: iOS and Android – Cross Platform Challenges and Solutions. ACM Digital Library. 2012. https://doi.org/10.1145/2341910.2341913 (01.01.2018)

[3] Messaoudi F., Ksentini A., Simon G., Bertin P. Performance Analysis of Game Engines on Mobile and Fixed Devices. ACM Digital Library. 2017. https://doi.org/10.1145/3115934 (01.01.2018)

[4] Michael H. Epic Games shows off the cross-platform Unreal Engine 4 with a Flappy Bird clone. 2014. https://www.phonearena.com/news/Epic-Games-shows-off-the-cross-platform-Unreal-Engine-4-with...-a-Flappy-Bird-clone_id56401 (02.01.2018)

[5] Bayliss D. Java For Android. 2015. https://www.raywenderlich.com/110452/java-for-android (02.01.2018)

[6] Rich, Francisco A. The Unique Weapons Of Ancient Japan As Used By Samurai, Ninja, Warrior Monks, Ruffians And All-around Badasses. 2015. https://www.tofugu.com/japan/ancient-japanese-weapons/ (17.01.2018)

[7] Schell J. The Art of Game Design – A Book of Lenses. United States of America, Burlington: Morgan Kaufmann Publishers. 2008.

[8] Khronos Group. OpenGL ES Version 3.2. 2016. https://www.khronos.org/registry/OpenGL/specs/es/3.2/es_spec_3.2.pdf (06.05.2018)

[9] Kessenich J., Baldwin D., Rost R. The OpenGL Shading Language. 2017. https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.4.60.pdf (06.05.2018)

[10] Ericson C. Real-Time Collision Detection. United States of America, San Francisco: Morgan Kaufmann Publishers. 2005.

[11] Westenberg J. Do you prefer physical or on-screen navigation buttons? 2017. https://www.androidauthority.com/physical-on-screen-buttons-poll-761259/ (19.04.2018)

Appendices

I. Glossary

Android	An operating system mostly for mobile devices, for example smartphones and tablets[22].
Shuriken	Commonly known under the names of throwing, ninja, or Chinese stars. Can be translated from Japanese as "hand-hidden blade". Although they can have many different shapes and sizes, the classic shuriken has multiple points, spins in flight and therefore does not require as much skill to throw as throwing knives of other shapes. [6]
Fragment	In computer graphics, it is the data, which is required in the process of generating a single pixel in the framebuffer[23].
Vertex	In computer graphics, it is a data structure containing attributes for a point in 2D or 3D space, for example position and colour[24].
Emulator	Software or hardware which lets a computer system imitate the work of another computer system[25].
Launcher	In the case of Android, it is an application which helps manage and launch other applications[26].

[22] https://en.wikipedia.org/wiki/Android_(operating_system)

[23] https://en.wikipedia.org/wiki/Fragment_(computer_graphics)

[24] https://en.wikipedia.org/wiki/Vertex_(computer_graphics)

[25] https://en.wikipedia.org/wiki/Emulator

[26] https://www.androidcentral.com/best-android-launchers

II. Installation Guide

Requirements for installing the game:

- Android version 4.0.3 or higher (confirmed to work on 4.1.2 or higher)
- ~12 MB of free storage
- ~30 MB of free RAM

Please follow the instructions listed below to install *Shuriken Way* onto an Android device:

I. Locate the installation package (*shuriken-way.apk*) from the Accompanying Files (Appendix IV).
II. Install a file managing application from the Google Play store to the Android device. For example ZArchiver[27], which was used during the work for the present thesis.
III. Continue with step 4 to get the installation package onto the Android device using a USB cable or skip to step 7 if you already have the installation package on the device.
IV. Connect the Android device to the computer that the installation package is on with a USB cable compatible with the Android device.
V. Set up the device and the computer for file transfer[28].
VI. Transfer the installation package (*.apk*) to a directory on the Android device.
VII. Locate the package on the Android device using the installed file managing application and run the file. Make sure not to accidentally view the contents of the file instead of running it.
VIII. Go to step 9 if the installation successfully begins. Otherwise, if installation from unknown sources is blocked, select "SETTINGS" on the displayed notification, which opens the needed page of the settings. Find and toggle the setting which allows installation from unknown sources. It is most likely called "Unknown sources". Go back to step 7.
IX. Wait for the application to install.
X. Locate the installed application on the home screen or in the "all apps" menu and run it.

[27] https://play.google.com/store/apps/details?id=ru.zdevs.zarchiver

[28] https://support.google.com/nexus/answer/2840804?hl=en

III. Game Guide

To start playing Shuriken Way, the game should first be installed onto an Android device (Appendix II). Once this is done and the game is started, there should be the menu screen displayed with the game title at the top.

There should also be numbered buttons right below the game title. The numbers represent the levels. You can always come back to this menu while playing a level by pressing the back navigational button on the device. Start from the first level by pressing the button with the number 1 on it (Illustration 1).

Illustration 1. Starting the level.

The first level consists of 3 simple platforms and 2 coins. The level is started with the shuriken attached to a platform. Tap the screen to detach from the platform (Illustration 2). The shuriken will jump in the direction which is away from the platform's center.

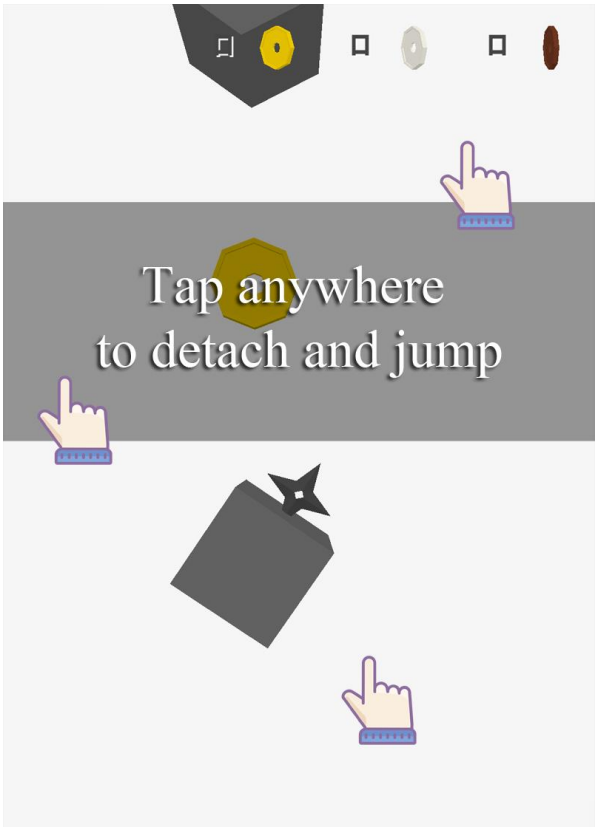

Illustration 2. Moving around in the level.

The gravity will pull the shuriken down. Falling too far will end the level and it can be restarted by pressing the button 1 again. This time wait for the center of the shuriken relative to the platform's center to point in the direction of the first coin and the next platform. If you wait for too long, the coin will drop in value and become silver and then copper. The level can still be successfully finished though. This will only affect the final score for the level. After collecting the first coin and landing on the second platform repeat the same for the second coin and the third platform. In most levels, the shuriken needs to land on a platform after all coins have been collected to successfully complete the level. After the first level is completed try completing the second level and then go to the third level. Some levels in *Shuriken Way* might be very challenging.

During the third level you will encounter a new object type, which is the accelerator. Hitting the spherical part of the accelerator will increase the speed of the shuriken in the direction that the arrow of the accelerator is pointing to. Try playing level 4 to see what happens when multiple accelerators are placed in a long chain. Levels 5 to 12 will introduce even more new game objects, which are the wall, the active wall, the relativity platform and the katana.

IV. Accompanying Files

The archive file containing the accompanying files has the following structure:

- /Source – the folder containing the source code files of the game
- shuriken-way.apk – the installation package for the game
- questionnaire_data.csv – the file containing raw data from the responses to the questionnaire
- demo.mp4 – a simple video demonstration of the gameplay

V. Graphics of Ninja Star!

With the help of Graphics API Debugger tool, it can easily be confirmed that *Ninja Star!* is, in fact, rendering two-dimensional textured polygons to represent the arena and various objects.

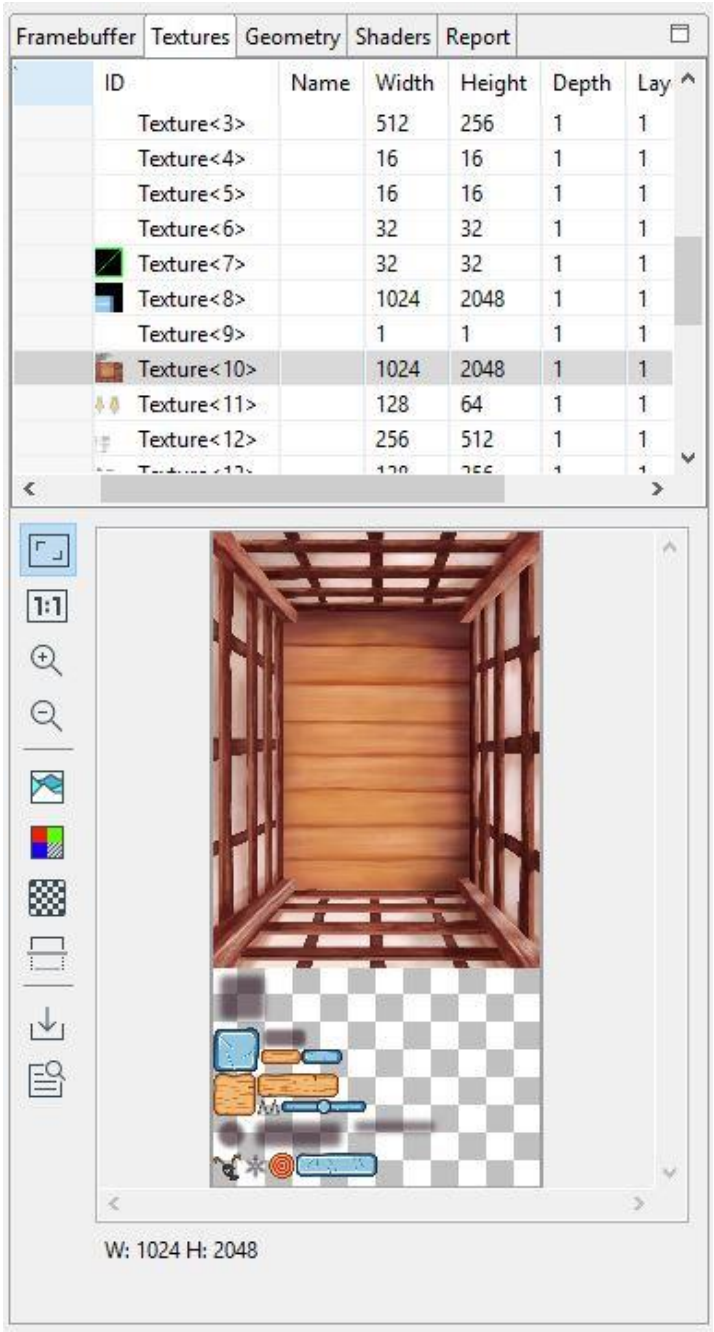

Illustration 1. Textures view in GAPID.

As can be seen in the screenshot above, GAPID can be used to view the textures which are accessible by the OpenGL at a certain point in time (Illustration 1). Selecting a command call from the command calls list defines the point in time as right after the command had

been executed. For example, the two screenshot below shows the *glDrawElements* command being selected (Illustration 2). Examining the parameters used in the command *glDrawElements* and preceding command calls tells us that 2 triangles are being drawn using the texture with identifier 10. As can be seen in the Illustration 1, the texture with identifier 10 is the texture that contains the images of the arena and some game objects. After the *glDrawElements* command had been executed, the previously empty framebuffer was filled with the image of the arena (Illustration 3). Every other game objects is drawn in a similar way.

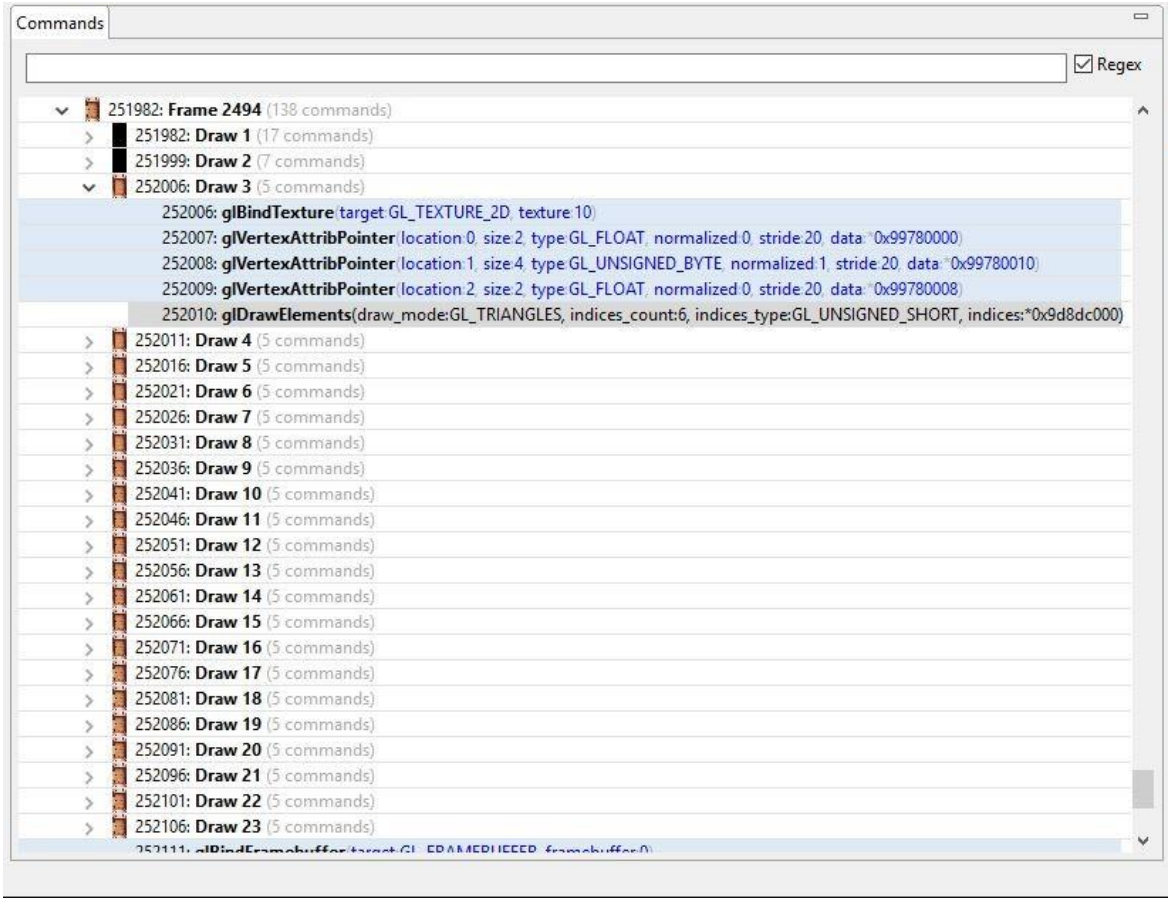

Illustration 2. A call of command *glDrawElements* selected in the Commands view of GAPID.

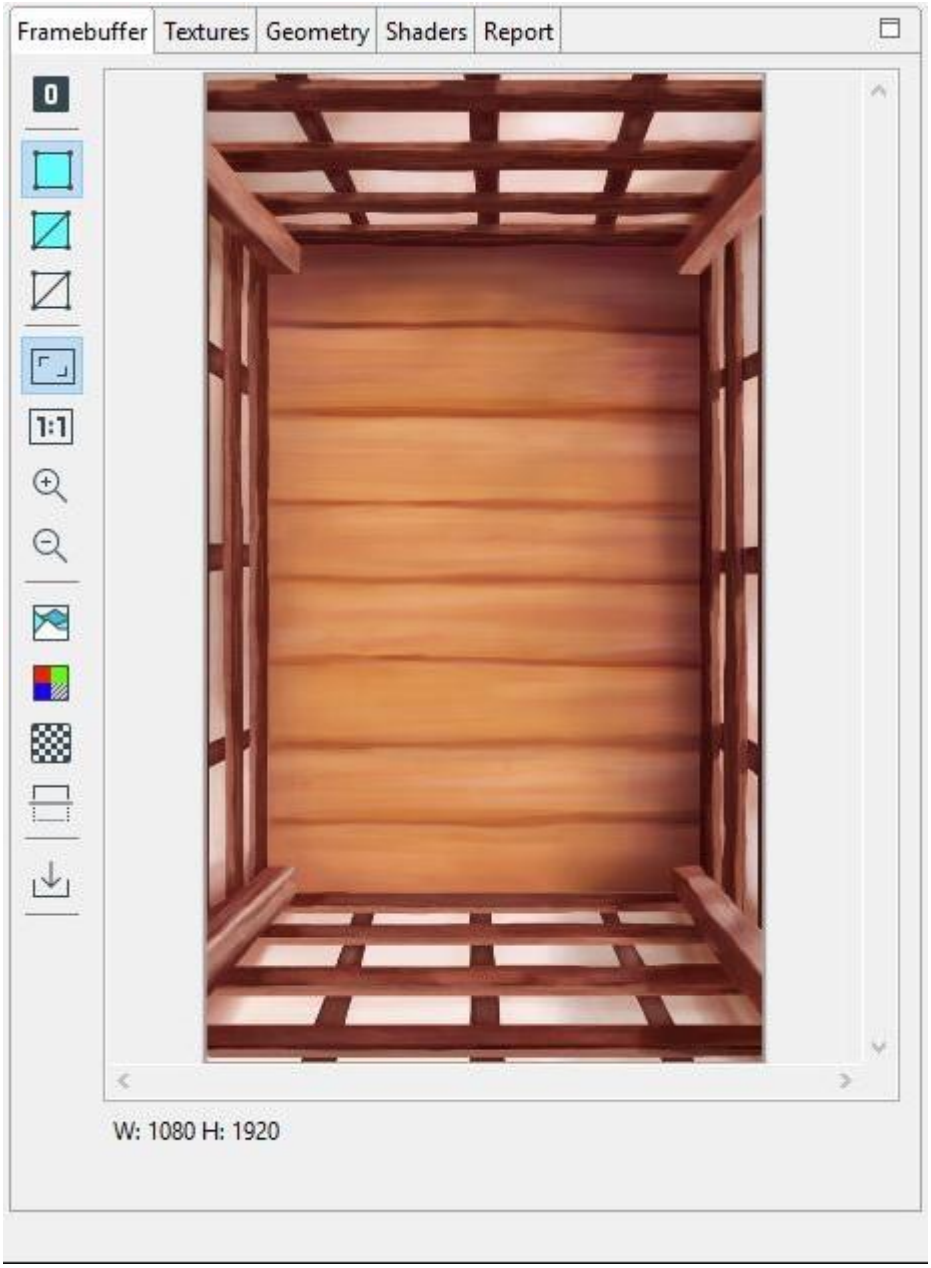

Illustration 3. Framebuffer view of GAPID showing the framebuffer filled with the game background.

VI. Unity vs Other Approaches

A total of 5 random Android games developed with Unity and 5 random Android games developed using alternative methods have been chosen and, with the help of Graphics API Debugger tool, average frame times for both approaches have been calculated. The debugging process slows the graphics down by a factor so the calculated frame times are often higher than what they would be normally. The original LG Nexus 5[29] (2013) was used to run 30 tests. The following table contains the results of the tests (Table 1).

Table 1. The results of the 30 tests run.

Game	Unity?	Test 1 (5000 frames) (s)	Test 2 (10000 frames) (s)	Test 3 (10000 frames) (s)	Average frame time (s)
Rubik Cube	yes	158.961	495.712	443.713	0.0439
Doodle Bowling	yes	94.285	189.458	209.475	0.0197
Punch Club: Fights	yes	464.343	968.548	1031.26	0.0986
Viridi	yes	297.856	805.874	878.734	0.0793
Fallout Shelter	yes	505.393	1153.111	1161.478	0.1128
Average	**yes**	304.1676	722.5406	744.932	**0.0709**
Hocus	no	178.615	378.53	390.291	0.0379
Underhand	no	133.029	215.333	215.365	0.0225
Infinitode	no	132.841	223.569	299.618	0.0262
FortressTD	no	410.242	1000.6	981.345	0.0957
Trench Assault	no	340.929	743.511	771.671	0.0742
Average	**no**	239.1312	512.3086	531.658	**0.0513**

[29] https://www.gsmarena.com/lg_nexus_5-5705.php

The average frame time during the tests for games developed with Unity was 0.0709 seconds compared to the average frame time of 0.0513 seconds for other games, as can be seen in the last column of the Table 1. This means that Unity games were performing worse than other games on average. The test results are not statistically sufficient to prove that games developed with Unity are slower than other games. However, they illustrate the tendency of it being true. The opposite (that Unity games are at least as fast) cannot be concluded from the results, thus explaining why the author did not feel confident in choosing Unity as performance was a priority for Shuriken Way.

VII. Final Compatibility Tests

The following table contains the results of the performance measurements from the second set of compatibility tests (Table 1), which were conducted after implementing changes based on the results of the different tests and playtests.

Table 1. The performance measurement results from the second set of compatibility tests.

Device	Screen resolution	Average frametime (ms)	Average update time (ms)	Noticeable visual stuttering
LG Nexus 5	1080 × 1920	3.1	0.07	No
Huawei MediaPad T3 10	800 × 1280	3.6	0.08	No
Xiaomi Redmi Note 4	1080 × 1920	15.1	0.11	No
Samsung Galaxy Grand Prime	540 × 960	2.7	0.03	No
OnePlus 3	1080 × 1920	13.7	0.2	No
OnePlus 5T	1080 × 2160	14.5	0.2	No
LG G4c	720 × 1280	6.6	0.57	No

No new compatibility issues were discovered during the second set of compatibility tests.

VIII. Applications for Graphics Performance Measurement

The following is a list of applications for Android that claim to measure graphics performance of games which were found during the work for the thesis:

- *FPS Meter*[30]
- *GameBench*[31]
- *Game Booster*[32]
- *GLTools*[33]

FPS Meter worked as claimed but it was not suited for the needed task. This is because it only displayed the current framerate on the screen without logging the displayed values. *GameBench* only allowed 30 minutes of monthly testing time for free, which was not enough for the needed task. *Game Booster* did not seem to display the correct framerate as it was showing approximately 59 at all times, even during obvious stuttering. In contrast, *FPS Meter* displayed a low framerate during stuttering. *GLTools* was not a free application.

[30] https://play.google.com/store/apps/details?id=com.ftpie.fpsmeter

[31] https://play.google.com/store/apps/details?id=com.gamebench.metricscollector

[32] https://play.google.com/store/apps/details?id=com.burakgon.gamebooster3

[33] https://play.google.com/store/apps/details?id=com.n0n3m4.gltools

IX. Full Results of the Performance Comparison

The following table contains the full table of results for the conducted performance comparison tests described in the subchapter 6.2 of the present thesis (Table 1).

Table 1. The full table of results for the conducted performance comparison tests.

Test	**Number of frames**	**Time (s) for *Shuriken Way***	**Time (s) for *Ninja Star!***	**Time (s) for *Shuriken***
1	5000	114.123	122.168	119.608
2	5000	122.75	111.138	106.428
3	5000	111.132	118.018	97.855
Average for 5000 frames		116.002	117.108	107.964
Test	**Number of frames**	**Time (s) for *Shuriken Way***	**Time (s) for *Ninja Star!***	**Time (s) for *Shuriken***
4	10000	260.957	230.329	267.783
5	10000	253.594	209.88	257.12
6	10000	261.237	235.144	233.954
7	10000	250.303	255.738	261.815
8	10000	267.534	265.548	260.984
9	10000	223.245	241.007	270.873
Average for 10000 frames		252.812	239.608	258.755
	Number of frames	**Time (s) for *Shuriken Way***	**Time (s) for *Ninja Star!***	**Time (s) for *Shuriken***
Total	75000	1864.875	1788.97	1876.42
Average frametime (ms)		**24.87**	**23.85**	**25.02**

The best performers on average in different categories are underlined in the table. *Shuriken* performed the best on average in the tests with 5000 frames. *Ninja Star!* performed the best on average in the tests with 10000 frames. *Shuriken Way* took the second place in both types of tests and in total.

X. The Questionnaire For Playtests

The following images are the screenshots of the questionnaire used in the playtesting part of the testing stage described in the subchapter 6.3 of the thesis (Illustration 1, 2 and 3).

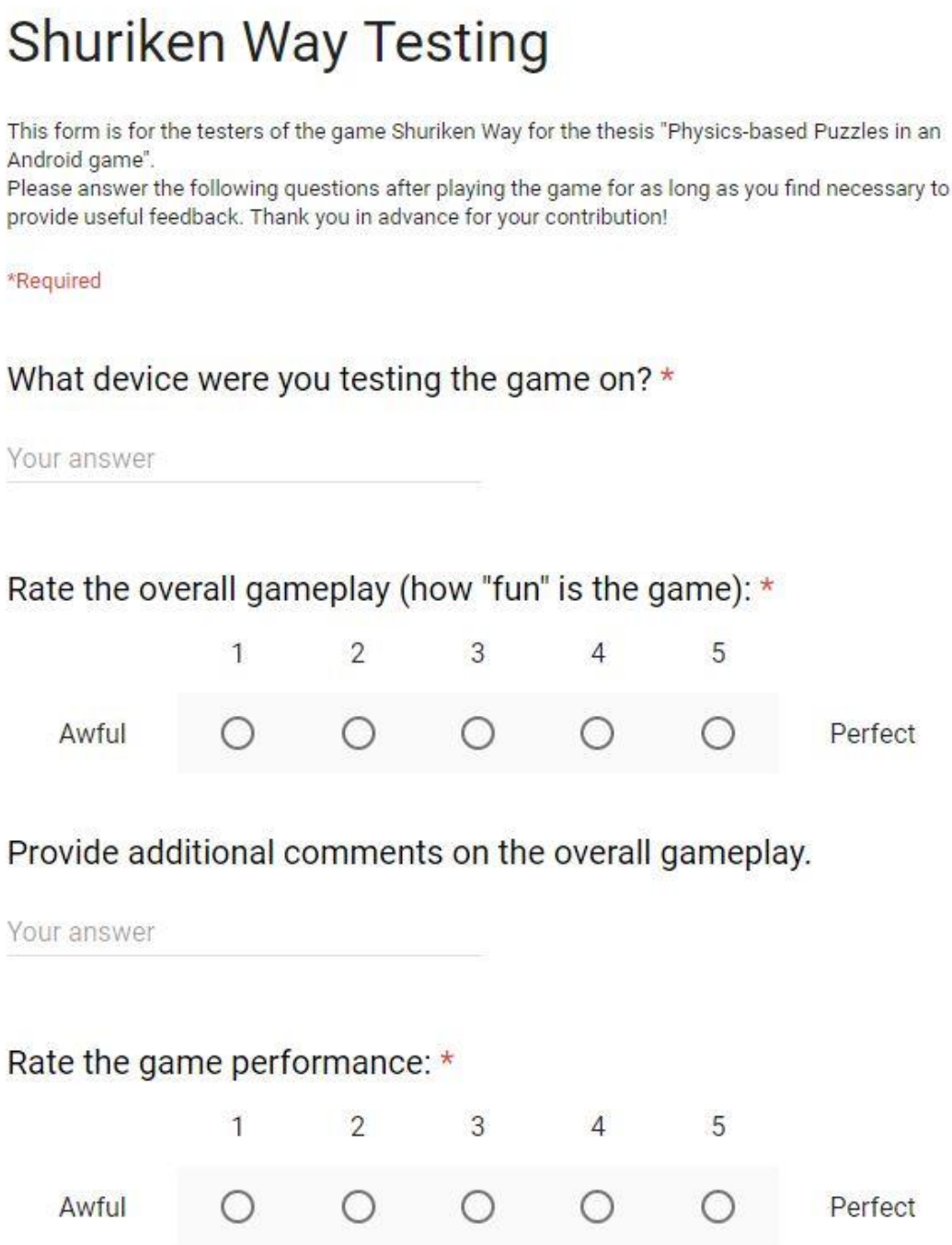

Illustration 1. First screenshot of the questionnaire.

Provide additional comments on the game performance:

Your answer

Were the controls confusing? *

- ◯ Yes, I'm still confused.
- ◯ Yes, but only in the beginning.
- ◯ No, everything was clear right away.

What else was confusing in the game if anything?

Your answer

Rate the level balance (how well is the difficulty of levels ordered): *

	1	2	3	4	5	
Awful	◯	◯	◯	◯	◯	Perfect

Suggestions to improve the level balance (reordering the levels or modifying them):

Your answer

Which game mechanics did you NOT enjoy?

- ☐ Platform rotation
- ☐ Gravity
- ☐ Accelerators

Illustration 2. Second screenshot of the questionnaire.

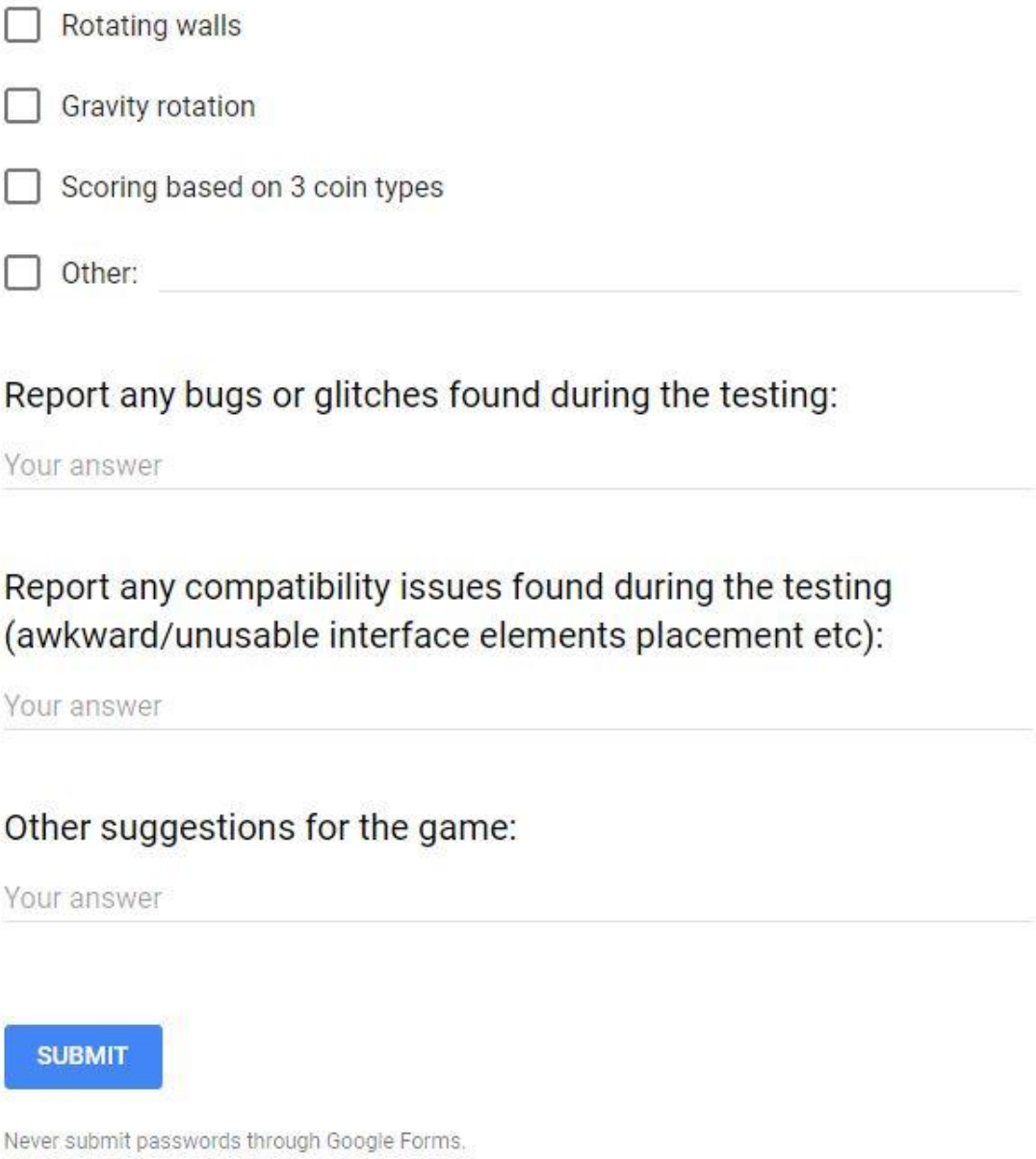

Illustration 3. Third screenshot of the questionnaire.

XI. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Anton Tšugunov**,

(*author's name*)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:
   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and
   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Shuriken Way – An Android Puzzle Game,

(title of thesis)

supervised by Raimond-Hendrik Tunnel,

(supervisor's name)

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **13.05.2018**