*ran dom*

*RUN*

(by HErC)

Game Development Document

1. Introduction

Random Run is an endless runner game written in Java using the LibGDX framework. The goal of the game is to have Blaze, the protagonist, survive for the longest period of time without colliding with obstacles, some of which can be broken by the adventurer’s attack.

A close up of a sign

Description automatically generated

Image 1: Random Run screenshot

1. Concept

The basic idea for the game was to create a simple, retro-style endless runner with easy-to-master control mechanics that offered a bit more variety into the genre. In addition to the regular “jumping to avoid obstacles” gameplay, slide and attack controls were added to the mix.

The sources of inspiration for the game were endless runners such as “Temple Run”, “Sonic Dash,” “Alto’s Adventure” and “Alto’s Odyssey”.

A close up of a sunset

Description automatically generated

Image: Alto’s Adventure,

1. Gameplay/Control

In tune with the simple gameplay controls, only three actions are possible in-game: swipe/fling upwards to jump, swipe/fling downwards to slide, and tap to attack.

Specific obstacles are overcome in specific ways: Blaze must jump over fires, slide under clotheslines, and destroy posts using his sword.

1. Art Style

The chosen art style for the game reflects its overall simplicity, resorting to a 16-bit feel. The game art’s retro style was obtained from the website itch.io. The artists edermunizz, OcO, and rvros should be credit for their provision of the background, props, and adventurer sprites, respectively.

The background music was obtained at the website soundcloud.com. The composer Dead Seed must be credited for the song “Destiny”, which is the theme level’s theme.

“New” sound effects, unfortunately, were not located in a timely manner, and should be considered for purposes of audio feedback only. Some of the sources are unknown; the sounds for jumping, sliding, and obstacle breaking were taken from Sonic the Hedgehog CD, and the death sound was taken from Kid Chameleon, both SEGA games from the early nineties.

1. Screens

The game’s targeted simplicity is reflected in its screens, with only four different ones been implemented in-game: Splash Screen, Title Screen, Menu Screen and Play Screen. Functionality that could’ve been included in separate screens (such as Pause and Game Over Screens) were included in the Game Screen, to improve the overall flow. This is controlled by the level’s own states, which dictate what buttons and messages should be made available on screen. The current functionality of the Menu Screen refers mostly to the setting of the volumes for both background music and sound effects, meaning that the “Sensitivity” slider has no functionality implemented, even though its values can be read anywhere in the app.

1. Analysis

The project has been very instructive, having provided its programmer with the opportunity to better understand how Java, Memory Management, Android applications, and LibGDX work. LibGDX isn’t a very intuitive library at first, but it is very versatile and powerful, and allows the programmer to control various aspects of the game to be developed. Compared to an application developed in Unity, for example, LibGDX projects are very lightweight, and though its API isn’t as extensive, its similarity in format to the one available for the Java language itself makes it easy to understand the intended functionality of its classes and components.

One of the major issues experienced by the programmer was related to how the “model class,” ActorBeta, was hard to extend to something suitable to the game’s needs. As the book itself suggested, some of the functionality should’ve been moved to other classes, but due to time constraints, the current model was kept.

1. Known issues

This is a non-exhaustive list of know issues involving the game:

There are known glitches involving the background sprites, where the position is reset off-sync with the actual scrolling. Further tests will be performed to correct these issues.

The physics of the player character are somewhat floaty. Fine-tuning will involve adjusting the jump and gravity values.

Transitions between states may require improvement for consistency purposes. As the game is planned to have its speeds adjusted, transition times between animations will need to be reworked into a more dynamic combination.

The “sensitivity” button in the Menu Screen doesn’t have its functionality implemented yet, which means that the user’s commands produce the same response regardless of the intensity of the action (namely, the app doesn’t discern between a longer or a shorter swipe).

Obstacle positioning requires an improved algorithm, to allow for a better gameplay experience. Still on the topic of obstacles, the sprites for the clotheslines need to be reworked so as to allow the player to view the character actually sliding between the clothesline’s posts (the current format has the player sliding in front of them).

The game requires new, improved sound effects.

The game also requires an improved “timing,” to feel more “game-like” than “test-like.”

1. The future

As it is, the game still requires improvements. The code needs to be “de-spaghettified” and compartmentalized into proper factory and component classes. In the future, the programmer will consider using the same framework to develop other Android games. As stated before, the framework is vast and robust enough to support the development of lightweight apps. As for “Random Run,” the programmer plans to implement a powerup system (with pickups), improved sound effects, and more diverse obstacles. A more challenging mode may also be implemented, where players will accrue points for chaining actions into timed combos.