


Take Home Case	
JavaTakeHomeCase	
Periode Berlaku Semester Genap 2022/2023 <i>Valid on Even Semester Year 2022/2023</i>	Software Laboratory Center Assistant Recruitment 23-2

Materi

Material

- Object Oriented Programming
- JavaFX GUI Component
- JavaFX Layout Manager
- JavaFX CSS
- Event-Driven Programming
- Java Database Connection
- Design Pattern
- JavaFX 2D

Soal

Case

The Justice Courier

Jason loves retro-style arcade games. Their naturally challenging game inspires him to create a game called “**The Justice Courier**”, inspired by ninja-style retro games. What makes it a great game is its way of using **minimal graphics** resulting in a very great game even with **low resolution/pixel graphics**. You, as a junior developer, are participating in a training boot camp. Your assignment is to create a **simple clone** of the game called **The Justice Courier**. Below are the requirements of the game:

➤ Application

You must **build** the game mainly using **JavaFX**. You are **free to use** any of the **Java** or **JavaFX features** you **need** to build the game, but the use of **FXML** is **prohibited**. You are required to **implement** basic graphics rendering techniques such as **limiting** the render frame per second or **FPS**. You are **recommended** to implement a **game loops pattern** and some other features like

updating independent rendering to make sure that your game **runs** at a **fixed speed** among hardware with **different specifications**.

The game should be **full-screened** by **default** and must **adapt** to different **screen sizes**. The game **window** should **not be able** to be **resized**. Design the app such that the **layouts won't break** during **gameplay**.

➤ **Documentation**

Due to the complexity of the application, especially for junior developers, you are **required** to document **every single progress** you made in this project.

- a. **Submission date** (the **date** you **submit** the work)
- b. **Progress** or **Changes** made.
- c. Some **additional notes** related to the work (**if necessary**)

➤ **Software Design & Architecture**

For your app to run smoothly for various devices, you must apply **at least** one **optimization attempt** to make the app **run smoothly** or **more efficiently**. You can use game development practices like **Object Pooling** or Java practices like **Multi-Threading**, other practices are **allowed** as well. You also need to **design** the **application structure** so that it could be **easily managed**. You are **required** to implement the **MVC** (Model View Controller) **Pattern** for the **entire program structure**.

Besides the structural pattern, you are **required** to as well **implement** other **behavioral** and/or **creational patterns**. You are required to implement **at least 3** patterns:

- **Either State or Observer**
- Any 2 additional design patterns **as long they make sense and don't make your application unnecessarily complicated**.

Note: you **must** document **what design pattern** you use, the **reason** you use it, and the **class name** where you implement the pattern.

➤ **Assets and Database**

Due to the time it will take to prepare assets, especially if you haven't learned pixel art, many assets will be **available** for you to **use**. Make sure to use a **relative path** to access the file.

If you use a **fixed path**, your program will be **restricted** to access the **fixed path**. The assets are located in \assets folder with the **following structure**:

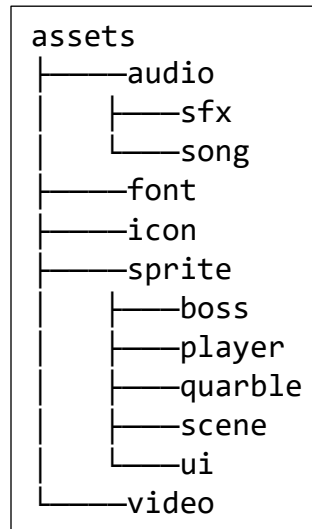


Figure 1. Assets directory structure

You are also provided with the **SQL script**. Run the SQL file located in \migration\create+insert.sql to **MySQL database** according to your **device environment**. For reference, the database table has the following **structure**:




HighScoreRecord	
 id	integer(10)
 name	varchar(20)
 time_spent	bigint(19)

Figure 2. HighScoreRecord table in TheJCourier database

Note: the **time_spent** field represents the **amount of time the** player spent defeating the boss in **milliseconds**.

➤ Main Menu



Figure 3. Main Menu

The title screen is the **first screen** the player will encounter upon opening the game. The title screen will consist of **three** main elements:

- **Background**

The background will display a piece of **resource image** that can be found in `\assets\sprite\main_menu.png`. **Simply display** the image and adjust it to **cover** the **whole screen**.

- **Title**

The **title** is a text with a **certain font** applied, it can be found in `\assets\font\prstartk-webfont.ttf`. **Place** the **title** at the **top center** of the x-axis and place it **slightly lower** than the top of the y-axis.

- **Menu**

The menu section will consist of **four buttons** that will serve their functions. The buttons that are used in **this menu** and the others **throughout the program** will need to be **styled** as follow:

- **Border with 5px radius, and 3px thickness**

- **Transparent** border-color
- **Transparent** background
- No distinct **focused** or **pressed** stylings.
- Use a **certain font** located in `\assets\font\prstartk-webfont.ttf`

If a button has hovered, set **the border color** to **#fcdc80** then play the **cursor sound effect** located in the following (choose your preference):

- `\assets\audio\sfx\menu_cursor_8.wav` or
- `\assets\audio\sfx\menu_cursor_16.wav`

If the cursor **leaves** the **hovered button**, set the **border color** back to **transparent**.

Each button will have its **clicking sound**. All buttons in **Main Menu** will play the button **start sound** and while another button will play the **select sound**. Both sounds can be found at `\assets\audio\sfx\`. Each of those buttons will serve the below functionalities:

- **Play Button**
Redirect the player to the game scene and **start the game**.
- **High Score Button**
Open the high score menu and load all the data needed from the database.
- **Options Button**
Open the options menu and display its content.
- **Quit to Desktop Button**
Closes the game.

➤ Highscore Menu



Figure 4. High Score Menu

The high score menu will **display** the **5 best scores** that are recorded in the **database**. You will need to perform queries to the **public database** to obtain the data you need to display. Make sure you get the **top 5 data** with the **least** time_spent in the database. The time_spent (which is a bigint), will be **converted** to **00:00.000** format.

The main components of the menu will consist of the **base image frame**, which the resources can be found at \assets\sprite\ui, and a **TableView** or **GridPane** that has the following style:

- No Header
- No Border
- Transparent Background
- No distinct **focused** or **pressed** stylings.
- Use a **certain font** located in \assets\font\prstartk-webfont.ttf

It also has a **Back** button that simply navigates the player back to the **Main Menu**.

➤ Options Menu



Figure 5. Options Menu

The options menu will allow the player to **set the volume** for **music** and **sound effects**, set the option to **show the timer** during gameplay, and navigate to **Controls / Key Binding** page. The main component consists of two **image frames** (both are from the same image at `\assets\sprite\ui`), a **top menu**, and a **navigation menu**.

The **top menu** consists of **two slider** components, each corresponding to the **music** volume and **sound effects** volume, and a **check box** that corresponds to the **show timer** option. The **navigation menu** consists of two buttons that have specific uses. All components mentioned have the specifications and functions as follows:

- **Music Slider**

Has a range from **0 to 10 (inclusive)** with 1 as the gap between values, which determines the **background music volume** during **gameplay**. The slider has a distinct **bar** component for each value, which has **White** color when it's selected, and **Grey** color when it's not selected. When the player **holds click** the slider, **audio** from `\assets\audio\song\bgm_sample.mp3` will be **played in a loop** until the player

releases the slider. The **volume** of the played audio will **match** the **slider value**. Each value change will be **saved automatically**.

- **Sound FX Slider**

Has a range from **0 to 10 (inclusive)** with 1 as the gap between values, which determines the **background music volume** during **gameplay**. The slider has a distinct **bar** component for each value, which has **White** color when it's selected, and **Grey** color when it's not selected. If the **value** of the slider **changes**, it will play the **cursor sound effect** (located in \assets\audio\sfx\menu_cursor_8.wav) with the **volume** that **matches** the **slider value**. Each value change will be **saved automatically**.

- **Show Timer Checkbox**

Is **checked** by **default**. It will determine whether to **show or hide** the **timer** in the **gameplay scene**. Each value change will be **saved automatically**.

- **Controls Button**

Navigate to the **Controls / Key Binding** page.

- **Back Button**

Navigate back to the **Main Menu** page.

➤ Controls / Key Binding Menu



Figure 6. Key Binding Menu (in default setting)

The **Controls** menu will allow players to **change** their **key bindings** to their preferences. The key binding represents what key should be pressed for the player to **do certain actions** in the game. The **default setting** should be shown the first time when opening the game and can be **changed** while the game is still open.

The main component consists of **two image frames** (both are from the same image at `\assets\sprite\ui`), a **Key Binding Menu**, that contains a multiple set of **text** and **buttons**, placed in a **GridPane**, and a **navigation menu**.

To change the binding, the player can **press** one of the keys in the **Key Binding Menu**, then **type** the expected keys to change the key binding. The game then checks if the key binding is **allowed** to be used. Otherwise, **revert** it to the **previous setting**. The **allowed key bindings** are mentioned as follows:

- All alphabet keys.
- Arrow Keys
- CTRL
- ALT
- SHIFT
- ENTER

The remaining controls have specifications and functions mentioned as follows:

- **Default Button**
Revert the saved key binding back to the **default settings** (refer to **Figure 6** for the default key binding)
- **Back Button**
Back to the **Options Menu**.

➤ Pause Screen



Figure 7. Pause Screen

Pause Screen can be triggered during gameplay. It will **Pause All Animations** and **Timer** in the game scene. This **Pause Screen** always **stacks** above the **Game Scene** in a way that the **Game Scene** is still **visible** to the **players** (not pitch black). The main component consists of **two image frames** (both are from the same image at `\assets\sprite\ui`) and a **navigation menu**. Each of those buttons will serve the below functionalities:

- **Options Button**
Open the options menu and display its content.
- **Back to Game Button**
Return to the **game** (**resume** all the **paused animations** and **timers**)
- **Title Screen Button**
Navigate back to the **Main Menu**
- **Quit to Desktop Button**
Closes the game.

Keep in mind that this Option Menu behaves similarly to the **Pause Screen**. The **Game Scene** should still be **visible** to the **players**. While **switching pages**, the **previously opened page**

will be **closed** for performance. The example can be seen in **Figure 8** and **Figure 9**, where the **Options** page will be **closed** while we **open** the **Control** page.



Figure 8. Options menu during Gameplay

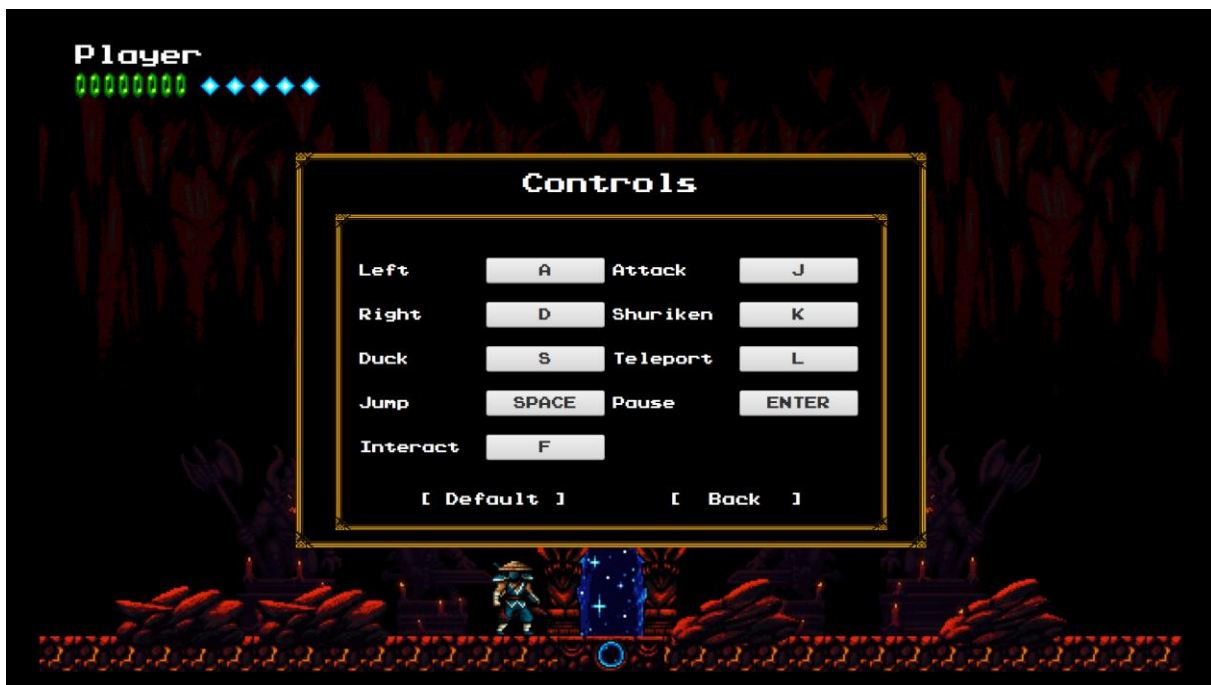


Figure 9. Controls Page during Gameplay

➤ Game Over Page

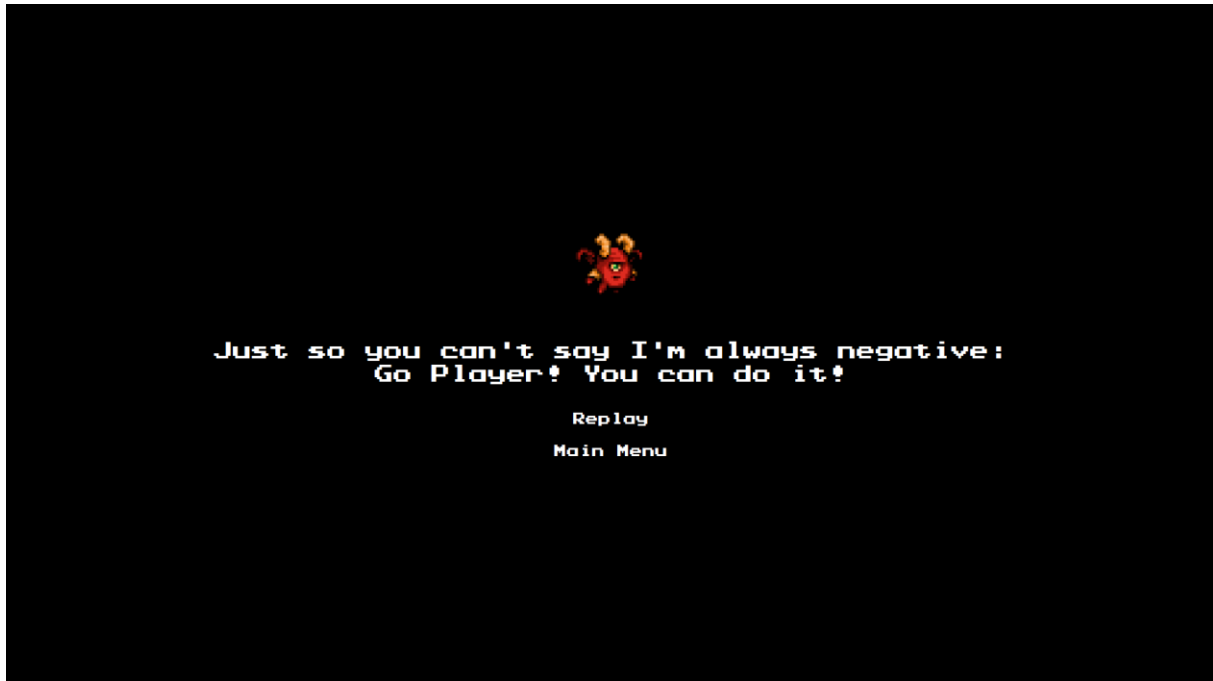


Figure 10. Motivation from Quarble

If the player's HP is set to 0, the player will **die**, and the user will be redirected to this page. It consists of 3 main components, an **animation** of flying Quarble, a "**motivation**" text (with **randomized** quotes, you can refer to [this page](#) for the quotes list), and a **navigation menu**. This menu will have two actions:

- **Replay Button**
Respawns the player back to the **Spawn Room**
- **Main Menu**
Navigate back to the **Main Menu**

➤ You Win Menu and Save High Score



Figure 11. You Win Screen

If the player **defeats** the boss, the player will receive a **popup** notifying that the **player wins** with the **time spent** to defeat the boss with the **00:00.000 format**. When the player **presses ENTER**, the player will navigate to **the Save High Score** page.



Figure 12. Save High Score Page

The main component of this page consists of an **image frame** (can be found in \assets\sprite\ui), a **text** to tell players to input their name, a **text box**, and **2 buttons**. Each of those components will serve the below functionalities:

- **Text Box**

This text box is a place to input the player's name. Add **validation** to check if the name **length** is **between 1 to 20 (inclusive)**.

- **Cancel Button**

Navigate back to **Main Menu** without saving the player's progress.

- **Submit Button**

Save the **player's name** and the **time spent** to defeat the boss to the database (refer to the **Assets and Database** section to see the **database details**). Make sure the player's name **validation** is checked first before **saving** the **player's highscore**. After the score is saved, navigate back to **the Main Menu**.

➤ **General Game Mechanics**

Due to the design nature of this game, there are some **rules of thumb** you must follow when building the **game scene**. The notable ones are **Resolution Control, FPS Control, Audio Control, Sorting Layer, Timer, and Pause Mechanic**. The details will be explained below:

- **Resolution Control**

The **game's resolution** must scale **accordingly to fit another device's screen**. But make sure the resolution **doesn't change** the **game layout** significantly (see **Figures 13** and **14** for example)



Figure 13. Spawn Room

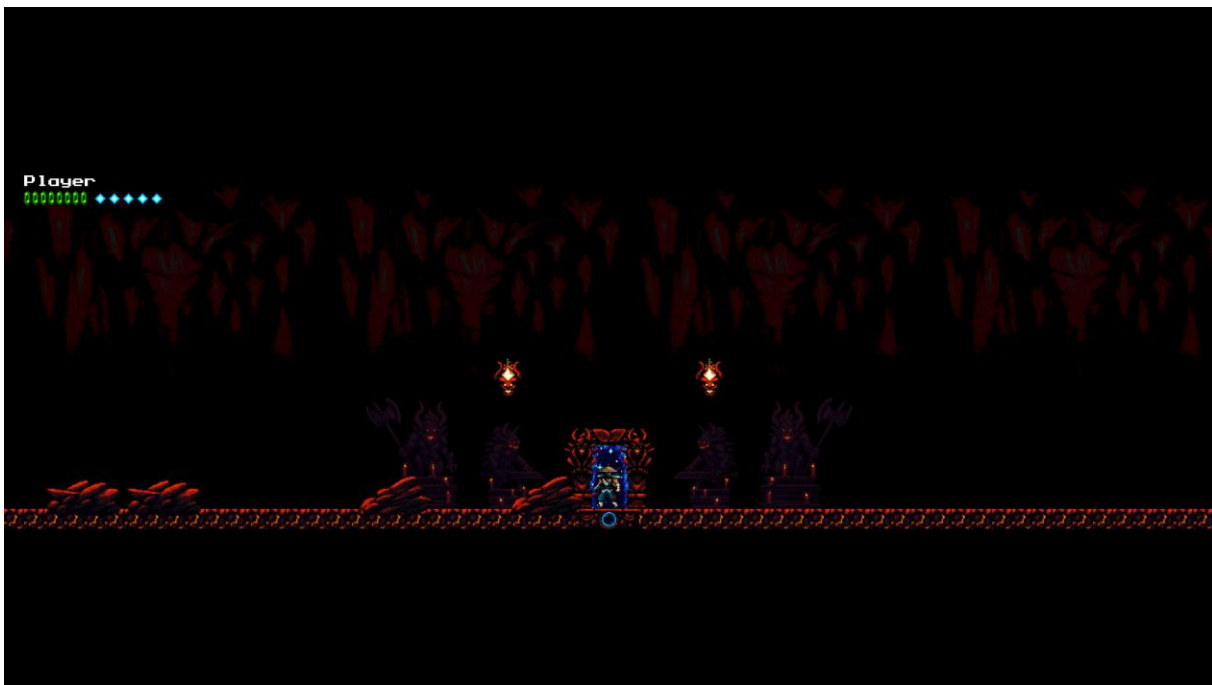


Figure 14. Spawn Room, but with bad resolution control

- **FPS Control**

Every device has a **different performance**, so it's possible the game will not **run smoothly** on some devices. Make sure to **control** your **refresh frame** (for example: **locked** at **60 FPS**) to **reduce screen tearing, synchronization errors, and other possible issues**. You should also make sure the player and boss move **at similar speeds** despite **running on different FPS**.

- **Audio Control**

In the game scenes, **music** will play in the **background**. Make sure the **music** and **sound effect volume** match the volume previously set in **Options Menu**. Also make sure the music plays a loop with the state mentioned in the following diagram:

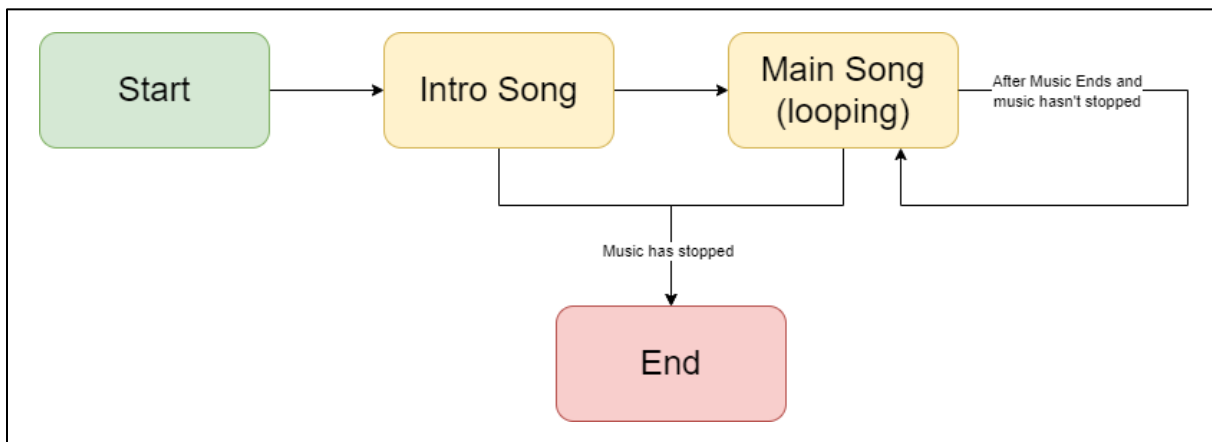


Figure 15. Music State Diagram

- **Sorting Layer**

Sorting Layer is a 2D **rendering mechanism** to determine which sprite should be rendered **first** before **others**. This is used to ensure some **desired objects** are rendered at a certain **position pattern** despite the scene **containing many objects**. Make sure to implement at the minimum **5 layers** for this game, which will be explained below:



Figure 16. Sorting Layer Illustration

1. Scene Background

This layer only consists of the **background image** used for the entire scene. You can find the background sprite in `\assets\sprite\scene\`. This background always renders **behind** all other objects in the game.

2. Background Layer (Object)

This layer consists of all background objects that always render **behind** the **Midground objects**. This layer consists of and is not limited to:

- Demon hive
- Gate
- Lantern
- Statues
- Throne

3. Midground Layer (Player)

This layer represents **all objects that can collide** by the player (**including** the **player** itself), that should be rendered **between Background** and **Foreground objects**. This layer consists of the **player** and the **boss**.

Note: You can consider **shuriken** and **player swing animation** as part of this layer too. But make sure it renders **in front of** both **player** and the **boss**.

4. **Foreground Layer** (Object)

This layer consists of all objects that always render **in front of** the **Midground objects**. This layer consists of and is not limited to:

- **Foreground Stones**

Example: a stone from sprite located in

`\assets\sprite\scene\Underworld_16_GroundAsset06.png`

5. **UI Layer**

This layer always renders in front of all other objects available in the game to prevent it from being obstructed by other objects. This layer consists of:

- **Player status** (including **health** and **shuriken count**)
- **Boss health bar**

- **Timer and Pause**

This game also features a timer that will count the player's time to defeat the boss. This timer is only active during **Boss Scene** and disabled during **Spawn Room** (the details will be explained separately in the **Boss Scene** section).

When pausing the game (the player presses the key bind to **pause**), the **Timer** will **temporarily** pause and stop the currently running background music, then a **Pause Screen** will appear (see **Pause Screen** section for details). If the player resumes the game, make sure the music volume **matches** the latest **volume** set by the player in the **Options Menu**.

➤ Spawn Room



Figure 17. Spawn Room

When the player presses **Play** in **Main Menu**, the player will first spawn in this scene. The actual purpose of this scene is for the player to test **animation**, **game mechanics**, and **Sorting Layer** (see **General Game Mechanics** section for details), including **player attack and movements**. This scene should have at least:

- **All 5 Sorting Layer Objects**
- **2 Lanterns**. These lanterns are **hittable objects** that will be useful for testing **player attack and movement** mechanics (see **Player Mechanics** for details)
- **1 Portal** (with **animated glow**).

When the player **interacts** with the portal (using key bind to interact), the player will **despawn**, then the scene will change to **Boss Scene**.

- **Timer UI** (**disabled** in this scene)

➤ Boss Room

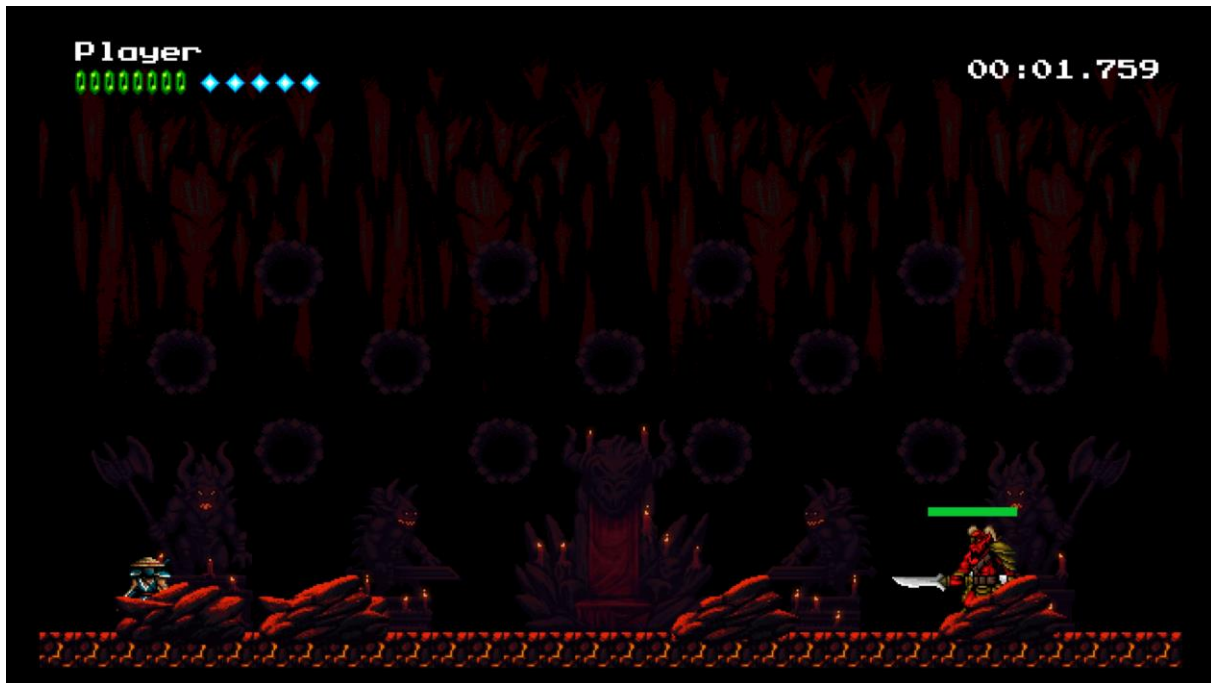


Figure 18. Boss Scene

The **Boss Room** is the room where the player will **fight** the boss. The boss fight starts 1.5 – 2 seconds after the player **spawns** in this room. The requirements for this page are:

- **The Boss.** Also put the **health bar** at the top of the boss (as a **UI Layer** object), so that the **health bar** will decrease as the **boss's health is reduced**.
- **13 Demon Hives** placed with **patterns** (see **Demon Hive Mechanics** section for details)
- **1 Animated Throne**
- **Timer UI automatically starts** as the player **spawned in**
- If the player **wins** the game (**defeats the boss**), **play** the **boss dead** animation while **pausing other object's animation (including the player)**, then show **You Win Menu** (see **You Win Menu** section for details)
- If the player **lost** the game (**health reaches 0**), play the **player's dead** animation while **pausing other object's animation (including the boss)**, then show **Game Over Page** (see **Game Over Page** section for details).

➤ Demon Hive Mechanics



Figure 19. Demon Hive sprite sheet

Demon Hive is a **hittable object** in **Background Layer** (see **Sorting Layer** for details) that serves as both **decoration** and an **advantage** for the player. It can be found in **Boss Room** that **complements** the **boss attack patterns** (see **Boss Mechanics** for details). Demon hive has **two states** that can be switched from player interaction “**showed**” and “**hidden**”. The **unique property** of the demon hive is if the player **hits one** of the demon hives in the scene (**Boss Room**), **all demon hives** in the same scene will **switch** their **state** (instead of affecting only that current demon hive).

In the **Boss Room**, the **demon hives** will spawn in a **pattern** to make sure the player is always possible to **cloudstep** by hitting **one** of the available **demon hives** during the **attack**. The pattern is mentioned below:



Figure 20. Demon Hive Tags

- Demon Hive in **tag A**, **tag B**, and **tag E** each has a 50% chance to start with **showed** state, otherwise **hidden**.
- Demon Hive in **tag C** has the **opposite** state as in **tag A** and **tag D** has the **opposite** state as in **tag B**.

➤ Player Mechanics



Figure 21. The player is in an idle state

Max Health: 8
Max Shuriken: 5
Shuriken cooldown: 10 second(s)
Abilities:
a. Melee
b. Shuriken
c. Glide
d. Cloudstep
e. Teleport

Figure 22. Player stats data

The ninja in **Figure 21** is a playable character in this game. This character has some **abilities** that will be used to defeat the boss, despite some abilities are **more complex** than others. The unique mechanic from the character is called **cloudstep**. If the player **masters** this technique, the player will be able to defeat enemies while **airborne** and **barely touching** the grass (**ground**). The character's movement sets are mentioned below:

- **Spawned**

This animation will play the **first time** the player **spawns** in a scene. After this animation, the player state will **automatically** turn into an **idle state**.

Sprite source: \assets\sprite\player\player_respawn.png

- **Died**

This animation will play when the player **dies** (when health **reaches 0** during the boss fight). After this animation **finishes**, the game will be **over**, and the player will be **redirected** to the **Game Over** page. Remember to play the **player died sound effect** too.

Sprite source: \assets\sprite\player\player_dead.png

- **Idle**

This animation will play while the player **doesn't move**. In this state, the **player's cape** will **flutter** in a stable motion.

Sprite source: \assets\sprite\player\player_idle.png

- **Duck**



Figure 23. The player in Duck State

This animation will play while the player presses the **duck** key bind. The player's height is also reduced to **match** the **player's height** (pay attention to the **green rectangle** that represents the **player's size** in **Figure 23**). The player also **can't move** or **attack** during this state. Although, the player can still **change direction** by holding the **left or right key bind**.

Sprite source: \assets\sprite\player\player_duck.png

- **Walk**

This animation will play **while** the player is pressing and holding either the **left** or **right** key bind. It will cause the player to **move** to the **left** or **right** at a **certain speed**. Make sure the speed is **consistent** despite the game running on **different FPS**.

When we pay attention to the **sprite source**, we can see that the sprite is split into 2 parts, the **top half**, and the **bottom half**. That's because the player can **attack while walking**, which will influence the **top half** animation, while the **bottom half** animation will **not change**.

Sprite source:

- \assets\sprite\player\player_walk_top.png
- \assets\sprite\player\player_walk_bottom.png
- \assets\sprite\player\player_walk_attack.png



Figure 24. Attack while walking

- **Jump (and Fall)**

This animation will **start** when the player **presses** and/or **holds** the **jump** key bind. It will cause the player to **jump**. This game also **applies gravity**, so the player can **fall** at some point. The **formula** to determine the **gravity** is mentioned below:

$$v_n = v_0 - a \times n$$

Figure 25. Velocity Formula

v_n = Velocity after n time passes

v_0 = The previous velocity

a = The fall acceleration speed (the value might vary)

n = the time range between the previous time (0) and current time (n)

When the player is **moving upward** (Jump speed > 0), play the **jump animation**, otherwise play the **fall animation** (when falling speed >= 0). These 2 animations will stop when the player **touches** the **ground**. Keep in mind that the player can also **attack while airborne** (while jumping or falling).



Figure 26. Attack while Airborne

Sprite source:

- \assets\sprite\player\player_jump.png
- \assets\sprite\player\player_fall.png
- \assets\sprite\player\player_attack_airborne.png

- **Hurt**

When a player **collides** with a boss or fire **hitbox**, the player will **receive damage**. If that happens, **reduce** the player's **health** by 1. Then the player will be **invincible** for **2 seconds** (unable to receive **additional damage**). While the player is **invincible**, the player's **rendering** will **blink every 0.1 seconds** (the player sprite will **switch** between **render** and **not render**). The player will also receive a **knockback effect** by adding some **force** to the **opposite direction** the player is facing, and some force **upwards**. The player's **cloudstep** also **actives** while hurt and still in **airborne**. This animation will **stop** when the player is **starting** other animations, but the **invincible** effect will **still work** while the player is **still invincible**.

Sprite source:

- \assets\sprite\player\player_hurt.png

- **Glide**

This animation will **only** trigger while the **player** is still **airborne**. **While** the player is **airborne**, the **cloudstep** is **not active**, and holding the **jump** key bind, the player will enter a **glide** state. The player's **fall speed** will **reduce** (this can be done by **limiting** the **fall speed** to a certain **cap**). Keep in mind that the player can **attack** while **gliding**. And the player will attack **downward** instead of **sideways** (left / right).

Sprite source:

- \assets\sprite\player\player_glider.png
- \assets\sprite\player\player_glider_attack.png



Figure 27. Attack while Gliding

- **Attack**

This animation will trigger when the player presses the **attack** key bind. The player can **attack** while in **another state** (except the **duck state**). Make sure that **1 swing** can only deal **1 damage** to some object (1 or more). The swing can only be triggered **once** per key press (if the player wants to **attack twice**, the player should press the **attack key bind twice**). The attack direction will **vary** depending on the player's state. If the player is **standing, walking**, or while in **airborne**, the attack swing will go **sideways** (left / right). If the player is **gliding**, the attack swing will go **downward** instead of sideways. This attack animation can hit any **hittable object** (including the boss). Doing so while in **airborne** will trigger **cloudstep**. If the attack **hits** the boss, the boss's health will **reduce** by **1**. Make sure to **play** the **sword swing sound effect** while attacking, and the **sword hit sound effect** if the **attack** manages to hit a **hittable target**.

Sprite source:

- \assets\sprite\player\player_attack_1.png
- \assets\sprite\player\player_attack_2.png
- \assets\sprite\player\player_attack_airborne.png
- \assets\sprite\player\player_glider_attack.png
- \assets\sprite\player\player_walk_attack.png

- **Shuriken**

The player can **throw shurikens** during gameplay by pressing the **shuriken** key bind. The player has **5 shurikens capacity**, which can be **refilled**. Every time the shuriken is **used**, it can be refilled every **10 seconds**, adding **1 shuriken** for the player to use again. The

shuriken will move **accordingly** to the current player's **direction** when the shuriken is used. When the shuriken **hits** the **boss** or **wall**, the shuriken will be **removed** from the scene. This shuriken can **deal 1 damage** if it **hits** the **boss**, but it **won't** trigger **cloudstep**.



Figure 28. Player shooting shuriken

- **Cloudstep**



Figure 29. Cloudstep indicator

Cloudstep is a **unique mechanic** that allows the player to **jump multiple times**. Originally, the player only can jump **once**. But if the player **hits** any **hittable object**, there will be a **cloudstep indicator** at the bottom of the **player's sprite**, and the player can **jump once more** while still **airborne**. This effect can be triggered **multiple times** as long as the player **doesn't touch the ground**. If the player touches the ground, the cloudstep effect **automatically expires**. This mechanism is **crucial** for this boss fight because there is an attack pattern that **relies** on this mechanic. If this mechanic doesn't work, the boss fights will be **very hard** if not **impossible to win**.

- **Teleport**

This skill allows the player to **teleport** to other positions at **short distances** according to the **player's position** (refer to **Figure 30**). You also need to apply **simple animation** at the **teleport destination** to enhance the player's experience (refer to **Figure 31**). This movement will be triggered by pressing the **teleport** key bind. The player can teleport **only once** when the player is **airborne**. If the player **jumps again** (from a **normal jump** or **cloudstep jump**), the player can **teleport again** (only once).



Figure 30. Player Teleport Destination



Figure 31. Simple Teleport Animation

➤ Boss Mechanics



Figure 32. Boss in Idle State

Health: 100

Boss states:

- a. Idle
- b. Dash
- c. Levitate

Figure 33. Boss stats data

The boss is the **primary antagonist** the player should fight in this game. The boss has **100 base health** with **2 attack patterns** with additional **animations**.

- **Health Bar**

To make sure the player has a **better understanding** of the boss's **current health**, add a **health bar** on top of the boss. It behaves as the **UI Layer** and contains 3 layers:

- **Red** (Represents health between **0 – 30 health**)
- **Yellow** (Represents health between **31 – 60 health**)
- **Green** (Represents health between **61 – 100 health**)

- **Idle**

In the beginning, the boss' first movement is **idle**. The boss stands still for **3 seconds**, playing **idling animation**. After the **3 seconds** have **passed**, the boss will **switch** to **attack moves**. Make sure the first attack movement is always **dash**. While the next animations will be either **dash** or **levitate** (feel free to use **any implementation** to decide the **next attack**, as long these **attacks** are used **equally**).

- **Dash**

After the first animation, the next and first attack move is a **dash**. There are **three phases** consists in this attack move:

- a. **Throw Sword (Dash)**



Figure 34. Throw Sword State

At the beginning of the attack move, the boss will **throw its sword** to the **left/right wall**, **according to the player's position** relative to the boss's position. After the **sword hits the wall**, the boss will **teleport to the wall** in a **climbing position**, then the boss will go to the next phase.

b. Dash



Figure 35. Dash state

In the second phase of the attack, the boss will **charge** for **0.5 seconds**. This charge animation is **indicated** by the **“blink”** of its **eye**. After the **charge phase** finishes, the boss will calculate its **movement direction** by **subtracting the last player's position** from the **boss' current position**. After that, the boss will move **according** to the **direction** until it **touches another wall**. This phase will occur **randomly at least 4 times**, but **not more than 10 times** before entering the next phase. Feel free to **adjust the random value range** to your preference (for example: 4 – 8 times is **still** a valid range). But make sure the **last two dashes** have a destination at the **top center** of the scene and the **bottom center** of the scene **respectively**. After the boss **finishes** the last dash (to the bottom center of the scene), the boss will **enter** the next and final phase.

c. Cooldown



Figure 36. Cooldown state

In this final phase, the boss enters its **cooldown state**. By playing **cooldown animation** for **5 seconds**. The boss will **stand still** at the **bottom center** of the scene and be **vulnerable** for the player to attack. After 5 seconds have passed, the boss will **go back** to an **idle state**.

- **Levitate**

In this attack move, the boss **will not** touch the ground and **levitate** instead. Because of how this attack move works, the player is also **forced** to **stay airborne** while attempting to attack the boss. Note that this attack move is **practically impossible** for the player if they cannot **cloudstep** properly. So, finishing the player movement is **mandatory** to finish this boss fight properly. This move consists of **two phases**:

a. **Throw Swords (Levitate)**



Figure 37. Throw Sword State

In this first phase, the boss will **throw two swords** both to the **left and right wall**. After the sword hits the wall, it will do two things:

1. The boss will **teleport** to either the **top left corner** or the **top right corner** of the screen.
2. A **fire** will spawn across two swords, practically making the **floor inaccessible** for the player without taking any damage.

Feel free to use any implementation to decide the boss teleport target position.

b. Levitate



Figure 38. Levitate Phase

This phase will last for **15 seconds**. During this phase, the boss will **stay still** in the **teleport position** (either in the **top left** or **top right** of the scene). Every **3.5 seconds**, the boss will **switch position** to the **opposite side** of the scene. Because the floor is **practically** inaccessible, the player must apply **cloudstep** by **attacking demon hives** in the scene to **stay airborne**. After 15 seconds have passed, the boss will teleport back to the **bottom center** of the scene and **go back** to an **idle state**.

c. Died Animation

If the **boss's health reaches 0**, the **boss will die**. The boss will **bounce slightly** to the **right** while gravity is applied to it with a reduced fall rate. The **floor collision doesn't apply** to the boss, so it will **move below** the current visible scene. You should also play the **boss_defeat sound effect** too.

d. Notes and Conclusion

The boss has a specific sound effect. Make sure to use it during some moves. For example, play the blink sound effect when the boss is blinking. Also, during the start of the boss fight, make sure to **hide** all **demon hives** during an **idle state**. When switching to any of the **attack states**,

make sure to refresh the **demon hives pattern**, then **show** all demon hives according to the **pattern**. After the attack pattern is **finished**, the **demon hives** should be **hidden again** until the **next attack phase** begins. In summary, the boss attack phases are **illustrated** in the diagram below:

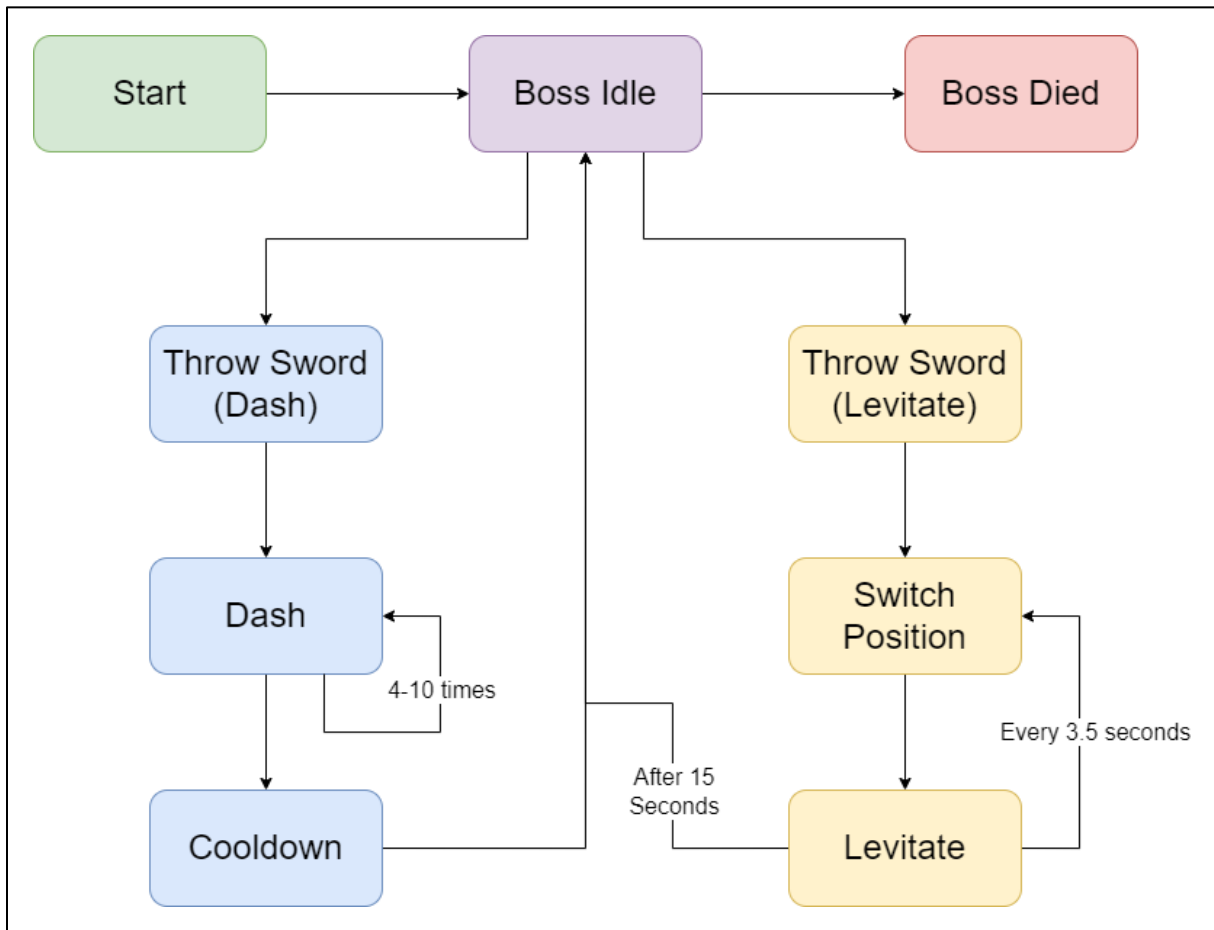


Figure 39. Boss State Diagram

Komponen Penilaian

Scoring Component

No.	Component	Percentage
1.	Java GUI and Menu(s)	24%
2.	Software Design & Architecture	8%
3.	JDBC	3%
4.	Java Canvas & Design	15%
5.	Player	25%
6.	Enemy (Boss & Demon Hive)	25%

Please run the BAT or JAR file to see the sample program.