Pixel Hackenbush

Seyed Mohammad Ghazanfari

January 18, 2024

Contents

Abstract

This repository contains a simple yet engaging implementation of the Hackenbush game, built using the Flutter framework and the Flame game engine. It's designed to be both fun and easy to play, leveraging the capabilities of Flutter for cross-platform compatibility and Flame for a smooth gaming experience [1].

[1] Clone this project from here.

Chapter 1

Introduction

Hackenbush is a combinatorial game with a set of rules and strategies that can be analyzed mathematically. The game is played on a drawn figure consisting of vertices and line segments connected to a base line called the ground. Players alternate turns, removing a line of their color—blue for the Left player and red for the Right player. When a line is disconnected from the ground, it and any lines depending on it are removed. Since this game is normal-play, the last player to make a move wins [1].

To understand more general positions in this game (like nimbers in Nim), We assign numbers to certain Hackenbush positions. We have the following equation for every Hackenbush position α:

$$\alpha + \bullet 0 \equiv \alpha \tag{1.1}$$

$\bullet 0$ is defined to be the Hackenbush position with no edges (so there are no available moves). The position $\bullet 0$ is type P. If α and β are Hackenbush positions, then

$$-(-\alpha) \equiv \alpha \tag{1.2}$$
$$\alpha + (-\alpha) \equiv \bullet 0 \tag{1.3}$$
$$\beta + (-\alpha) \equiv \bullet 0 \text{ implies } \alpha \equiv \beta \tag{1.4}$$

For every positive integer n, define $\bullet n$ to be the Hackenbush position consisting of n isolated blue edges. For a negative integer $-m$, we define $\bullet(-m)$ to be the Hackenbush position consisting of m isolated red edges. We may view $\bullet$ as an operation that takes an integer and outputs a special kind of Hackenbush position. For any integers m and n, we have

$$-(\bullet n) \equiv \bullet(-n) \tag{1.5}$$
$$(\bullet m) + (\bullet n) \equiv \bullet(m+n) \tag{1.6}$$

It is straightforward to determine the type of an Hackenbush position since in each such position some player has no moves.

$$\text{Type of position } \bullet n \text{ is } \begin{cases} L & n > 0, \\ P & n = 0, \\ R & n < 0. \end{cases} \tag{1.7}$$

We talked all about integer positions like $\bullet n$. Could there be a Hackenbush position α that gives a player an advantage of $\frac{1}{2}$? The answer is Yes! For every positive integer k, we have $\bullet\frac{1}{2^k} + \bullet\frac{1}{2^k} \equiv \bullet\frac{1}{2^{k-1}}$.

To generalize rules we talked about earlier, we turn to a special kinds of numbers called *Dyadic Numbers*. Any number that can be expressed as a fraction where the denominator is a power of 2 (and the numerator is an integer) is called a dyadic number. For example, $\frac{17}{32}$ and $\frac{531}{64}$ are dyadic numbers, but $\frac{2}{7}$ and π are not. It is obvious that any integer numbers are dyadic numbers (the denominator can be one or 2^0).

As every dyadic number has a unique (finite) binary expansion (can be proved), for every dyadic number $q > 0$ with binary expansion $2^{d_1} + 2^{d_2} + \cdots + 2^{d_l}$ (here $d_1 > d_2 > \cdots > d_l$ are integers which may be positive), we define the Hackenbush position $\bullet q$ as follows:

$$\bullet q = \bullet 2^{d_1} + \bullet 2^{d_2} + \cdots + \bullet 2^{d_l} \tag{1.8}$$

Chapter 2

Game Assets

The game's assets originate from external sources. To build and execute the project, you must independently acquire these assets by consulting the provided reference [2].

To utilize the images within the project, a conversion process is required where multiple images are combined into a single composite image. For instance, the main character's idle animation is composed of five separate images, each measuring 60×40. These should be merged horizontally to form a composite image with dimensions 320×40. Once you've transformed each set of animation frames into a single image, place them in the `assets/images` directory, ensuring they are sorted into the appropriate subdirectories. As an example, the main character's animations would be located in `assets/images/Character`.

To build and create a level for this project *Tiled Map Editor* is used. You can access this application by referring to [3].

Chapter 3

Gameplay

3.1 Menus

In this project, all menus are developed using the Tiled application. Each menu can be viewed in Appendix B. The design process is modular, allowing for the creation of complex menus using different properties. In other words, different properties are set to create complex menus using `flutter` and `flame_tiled` packages. For instance, you can create a button with different functionalities and types like icon button, toggle button, and more.

The `flame_tiled` package serves as a bridge between the Flame game engine and Tiled maps. It parses TMX (XML) files and accesses the tiles, objects, and other elements within them.

3.2 Player

In a game, each player is characterized by a 2D vector representing their velocity. This vector (x, y) indicates the direction in which the player can move. The game reads inputs from the keyboard, such as pressing arrow keys, enter, or space bar, and acts accordingly.

The game continuously checks the current state of the player and updates it if necessary. This could involve changing the player's velocity if they press an arrow key, updating the player's animation, like if they start or stop running.

3.3 Physics

Figure B.1 illustrates that each game object, including the player, walls, and ground, is represented by a rectangle box. The player class incorporates a `CollisionCallbacks` mixin, enabling the player to listen for collisions. Upon collision with another game object, the type of the object is assessed (e.g., `CollisionBlockType.platform`, and `CollisionBlockType.ground`), and actions are taken accordingly. Four methods are designed to determine the direction of the collision relative to the player.

To determine the collision direction, we calculate the normal vector between the player's position and the midpoint of two intersection points. This vector is then multiplied by predetermined vectors representing the four cardinal directions (`fromTop`, `fromBottom`, `fromLeft`, and `fromRight`). If the result-

ing value exceeds a certain threshold, the method returns `true`, indicating a collision in that direction. Otherwise, the player is not colliding with the object in that direction. Additional conditions ensure the player is colliding in a specific direction.

3.4 Interaction With Objects

In this modified version of the Hackenbush game, there are three players, each with the ability to attack a designated enemy:

- Player one can attack 
- Player two can target 
- Player three is assigned to attack 

Each player has a unique adversary and cannot attack enemies assigned to others. In Figure B.2 we can see an example of these enemies on top of each other.

When the player presses the attack button, a raycast is initiated from the player's forward direction to determine if an enemy is in range. If the raycast collides with an enemy that is compatible with the player's attack, that enemy and any additional enemies stacked with it are eliminated, in accordance with the Hackenbush game rules.

Appendix A

Assets Structure

The directory structure for `assets/images` should be organized as follows:

```
Backgrounds
  Additional Sky.png
  Additional Water.png
  BG Image.png
  Big Clouds.png
  Small Cloud 1.png
  Small Cloud 2.png
  Small Cloud 3.png
  Water Reflect Big 01.png
  Water Reflect Big 02.png
  Water Reflect Big 03.png
  Water Reflect Big 04.png
  Water Reflect Medium 01.png
  Water Reflect Medium 02.png
  Water Reflect Medium 03.png
  Water Reflect Medium 04.png
  Water Reflect Small 01.png
  Water Reflect Small 02.png
  Water Reflect Small 03.png
  Water Reflect Small 04.png
Character
  Air Attack 1.png
  Air Attack 2.png
  Attack 1.png
  Attack 2.png
  Attack 3.png
  Fall.png
  Fall Sword.png
  Ground.png
```

```
  Idle.png
  Idle Sword.png
  Jump.png
  Jump Sword.png
  Run.png
  Run Sword.png
Dust Particles
  Fall.png
  Jump.png
  Run.png
HUD
  A.png
  B.png
  Down.png
  Joystick.png
  Knob.png
  Left.png
  Right.png
  Up.png
  X.png
  Y.png
Terrains
  Platforms (32x32).png
  Terrain (32x32).png
  Terrain and Back Wall (32x32).png
Totems
  Head 1
    Hit 1.png
    Hit 2.png
    Idle 1.png
    Idle 2.png
  Head 2
    Hit 1.png
    Hit 2.png
    Idle 1.png
    Idle 2.png
  Head 3
    Hit 1.png
    Hit 2.png
    Idle 1.png
    Idle 2.png
UI
  Big Banner
  Big Text
```

Green Board
Green Button
Inventory
Life Bars
Mobile Buttons
Orange Paper
Prefabs
Sliders
Small Banner
Small Text
Yellow Board
Yellow Button
Yellow Paper

Appendix B

Images

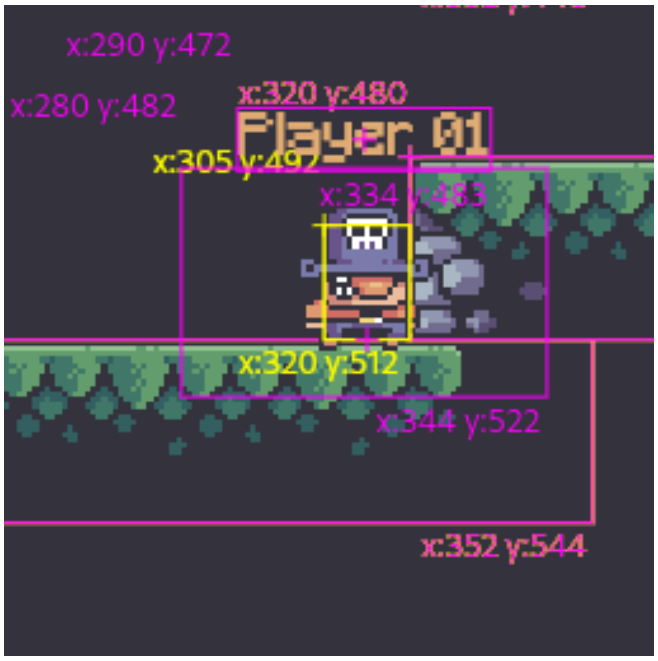

Figure B.1: Collisions

Figure B.2: Enemies

Figure B.3: Main Menu

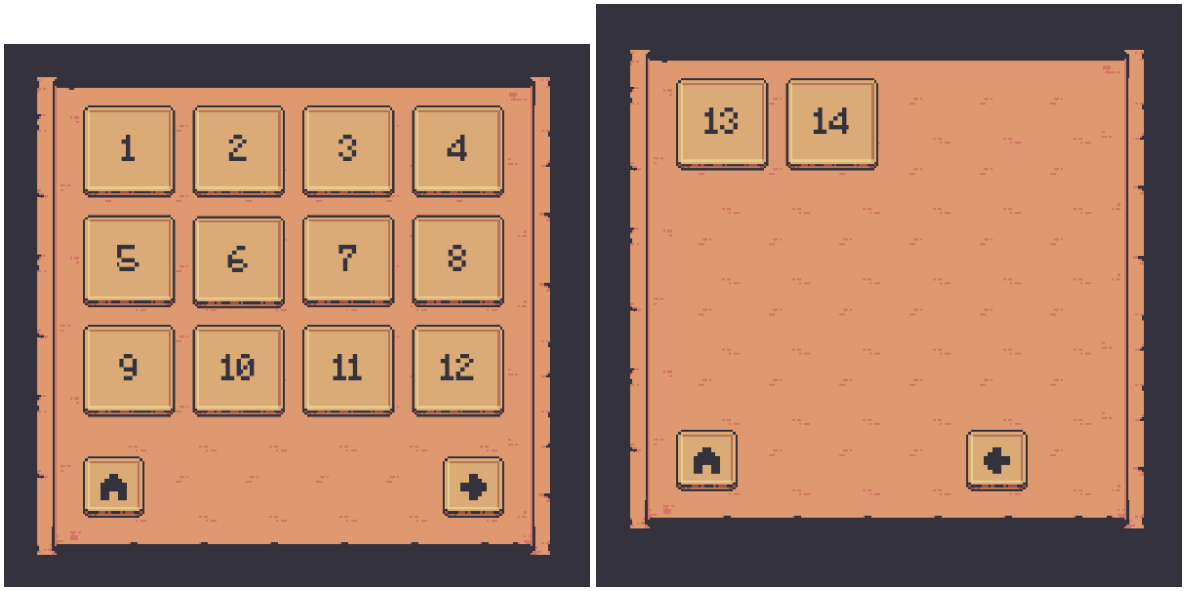

Figure B.4: Selecting Levels

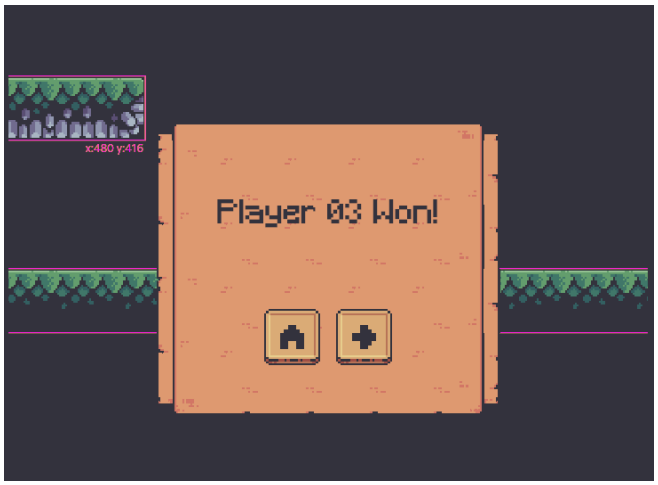

Figure B.5: Win Prompt

Bibliography

[1] Matt DeVos and Deborah A. Kent. *Game Theory: A Playful Introduction*. American Mathematical Society (AMS), 2016.

[2] Pixel Frog. Treasure hunters. https://pixelfrog-assets.itch.io/treasure-hunters, 2020.

[3] Thorbjørn Lindeijer. Tiled map editor. https://www.mapeditor.org, 2024.