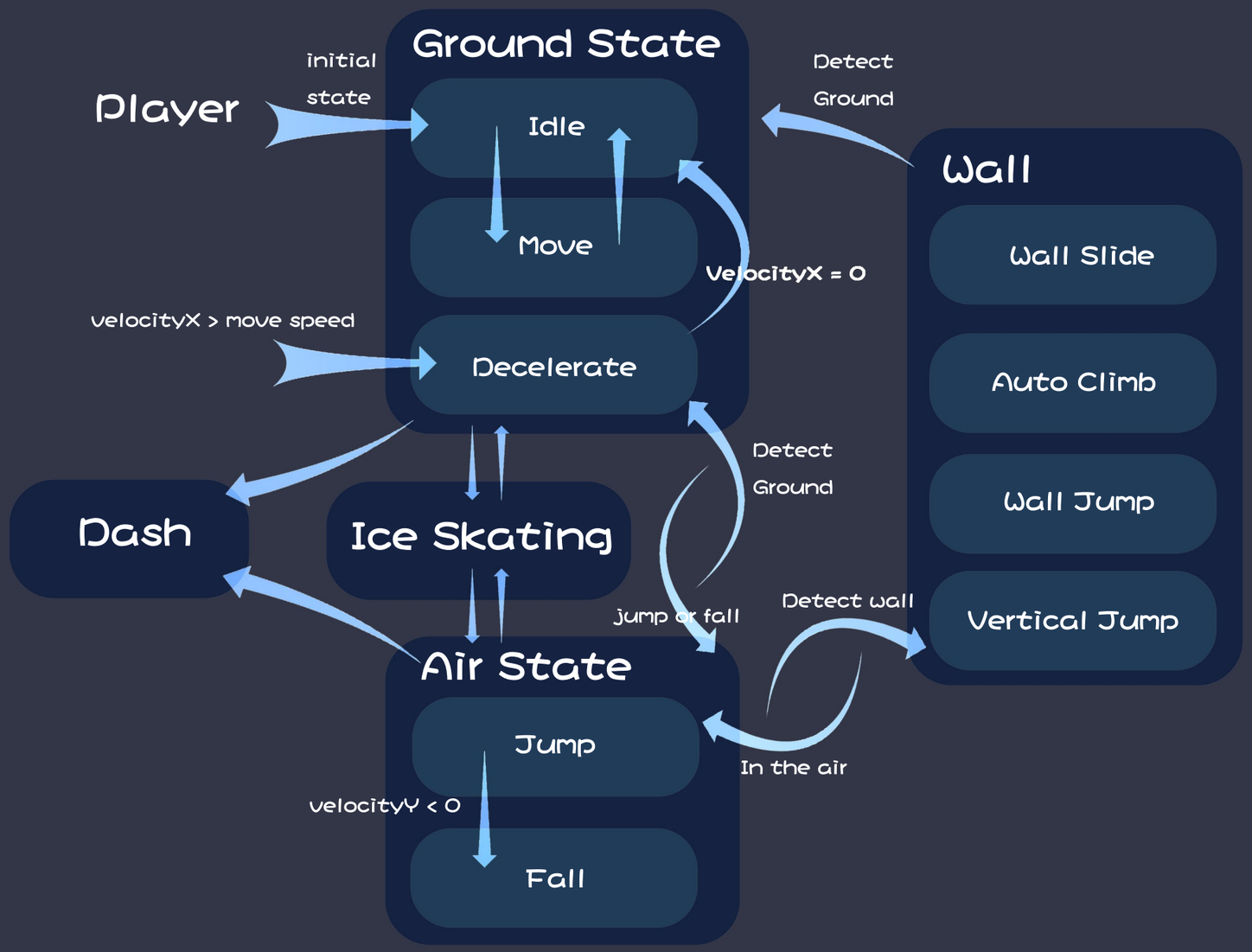
Prototype of 2D Platform Game

This project is a 2D game prototype framework made by Unity, aimed at optimizing character controls and game mechanics. I was deeply influenced by Celeste while creating this prototype.It serves as a foundation for the development of 2D platformers and Metroidvania games.

Script Code: https://github.com/XichaoWang97/2D-Platformer/tree/master

Download: https://pan.baidu.com/s/1DWLFOuFvkMzBHElJ6BQU8w (extraction code: uf4c)

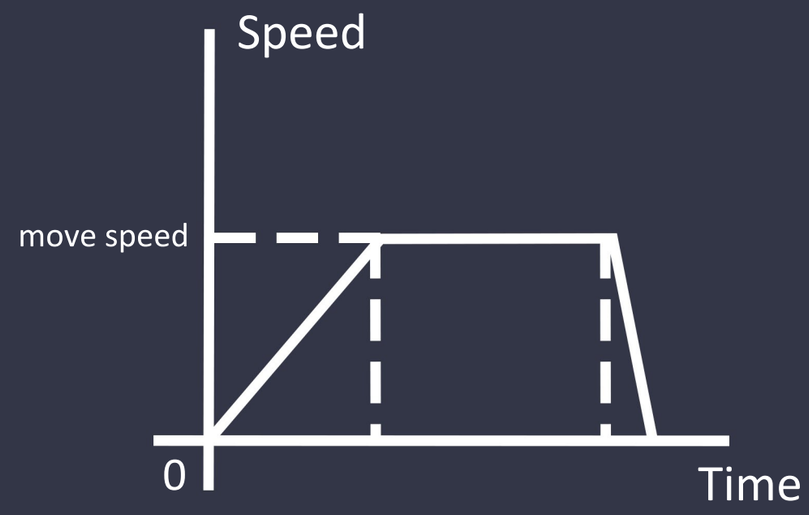
Player State Machine



Acceleration and Deceleration

To simulate more realistic movement, I added acceleration and deceleration mechanics to the movement system. The change in speed roughly follows the curve shown below:

The deceleration time is much shorter than the acceleration time, mainly to avoid "sliding". Cause having a long deceleration time can give players the feeling that it's hard to control .



Jump Buffer

If you press the jump button within a few frames before landing, the game should remember the command so that the character can still jump after landing, which will make the character jump more smoothly.

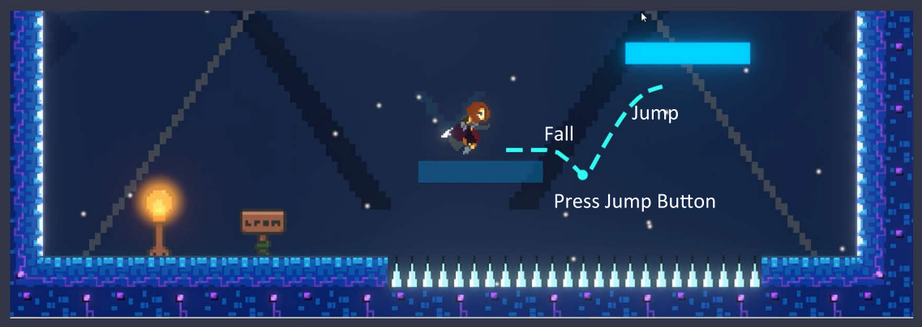
That means when the jump button is pressed, a frame countdown is enabled, and if the int variable is not 0 after landing, it will still return true, that is, the jump button is pressed.

Death Delay

When the player touches a trap, if they die immediately, it can cause a strong feeling of frustration. Therefore, I've added a brief death delay. If the player manages to escape the trap within this short delay, they won't die. Additionally, in some cases, if the player is moving fast enough, they can dash through the trap.

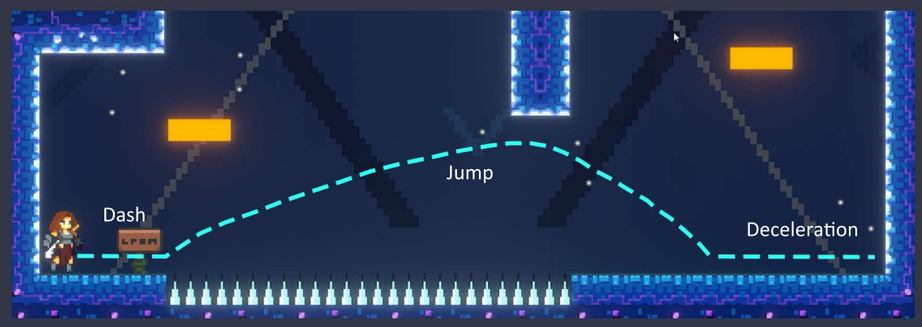
Coyote Time

After the character leaves a platform, there is a brief period during which the player can still jump.



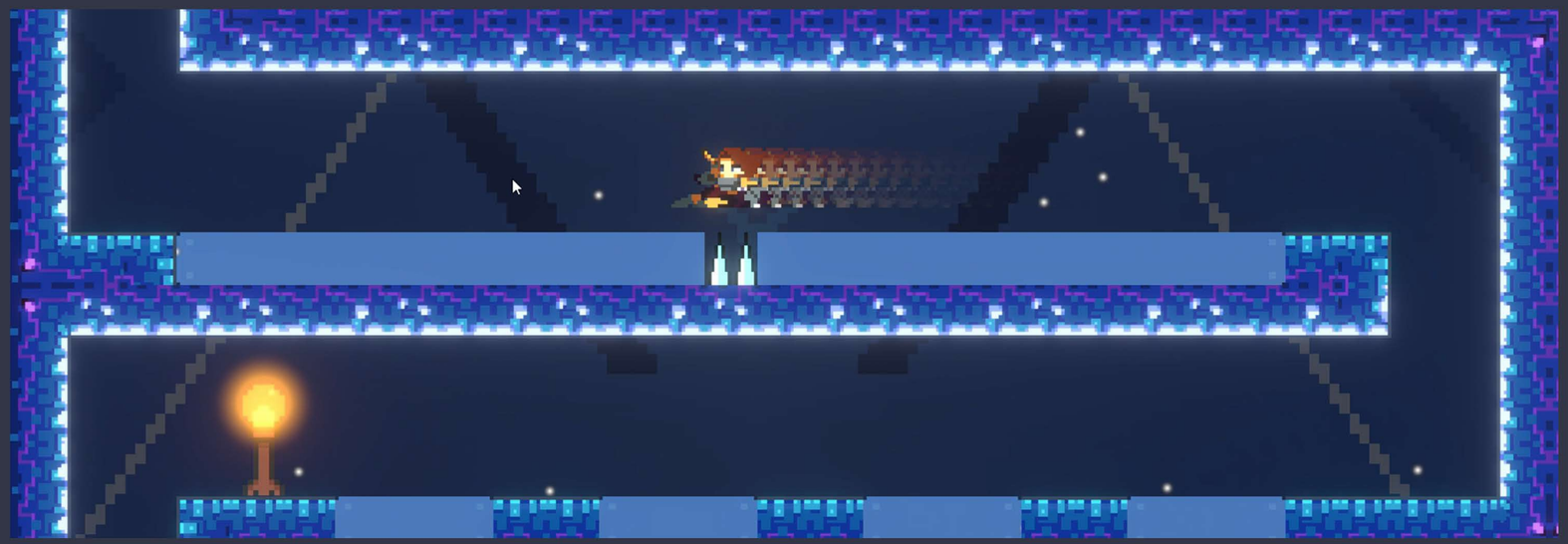
Dash Jump

Immediately pressing the jump key after dashing will make the character jump at a very high speed, covering a long distance.



The bright blue color lines in images illustrates the movement trace of the character.

Sliding



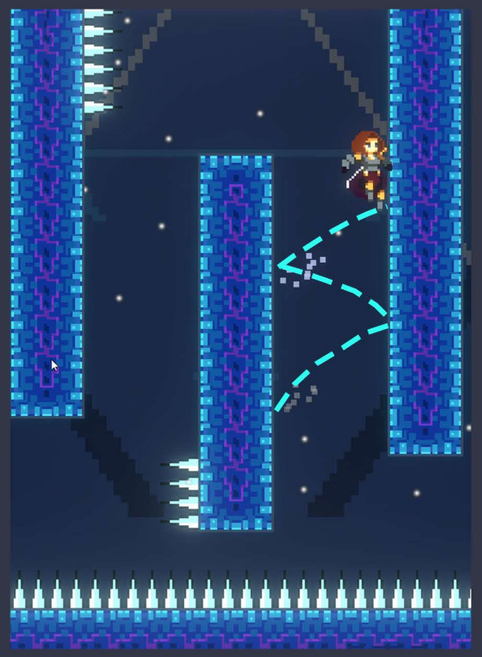
If player make contact with the ice surface, you will enter ice skating state. In this state, the player's speed is equal to their velocity of the moment they step onto the ice, but with a minimum value set to the player's move speed. This means that if the player enters the ice with a speed lower than their move speed, the skating speed will automatically increase to match the move speed.

interaction with death delay: when the player’s skating speed is fast enough, they can skate out of traps with-out triggering the death.

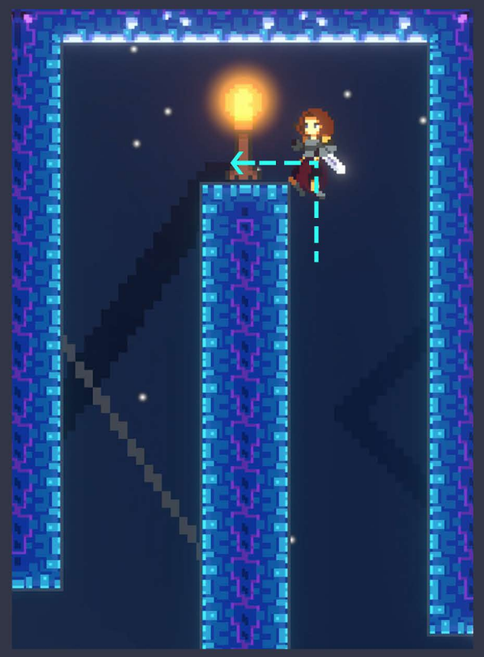
Climbing

There are some types of wall climbing situations:

- When the character has stamina, pressing the climb button will make the character stick to the wall.
- Dashing in the air, Jumping, and falling.



wall Jump



automatically move

Special Mechanisms about wall climbing

1.When having stamina, pressing vertical key and jump will cause a vertical jump.

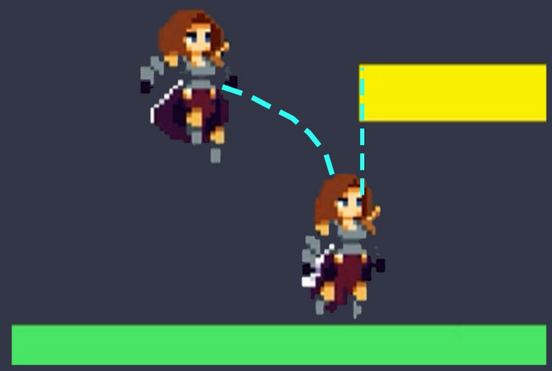
2.In the previous case, pressing only the jump key will cause a opposite direction jump

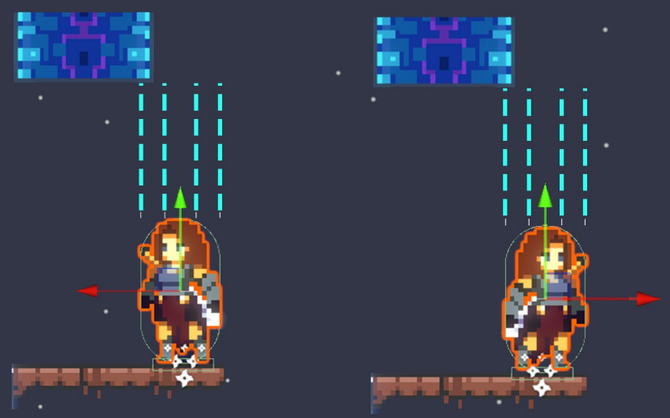
3.If the up button is pressed while on the edge of the wall, player will automatically be moved onto the platform.

About Collision

Based on Unity's physics simulation, the terrain sometimes obstructs the player's movement. While this might align with physical laws, it creates negative feedback for the player. For example, when using a box collider, the character's four corners may collide with the terrain and obstruct movement. Of course, most 2D games today use capsule collider as their characters' collider, which prevents the sharp corner collisions and instead results in rounded edge collisions.

My desire is as follows: If the player only collides with the terrain at a small area, the player should first perform a small displacement in the direction perpendicular to the movement axis before collision. Once the player moves to a position where no collision occurs, the displacement should stop, and the player could resume movement along the original axis.





To address this, I used raycasting to modify the behavior.
When the collision length at the player's edge is smaller than a certain value, the movement will proceed as desired.
In the picture, you can see the edge of terrain is between the two rays in one side.

For horizontal movement and falling with rounded edge collisions, the actual result aligns with the desired effect. However, when the character jumps and their head collides with the terrain (resulting in a rounded edge collision), displacement occurs, as shown in the image above, player is catapulted outside. This consequence does not match expectations.

Here is the consequence, player changes the locatation horizontally, without collision, and keep going upwards.