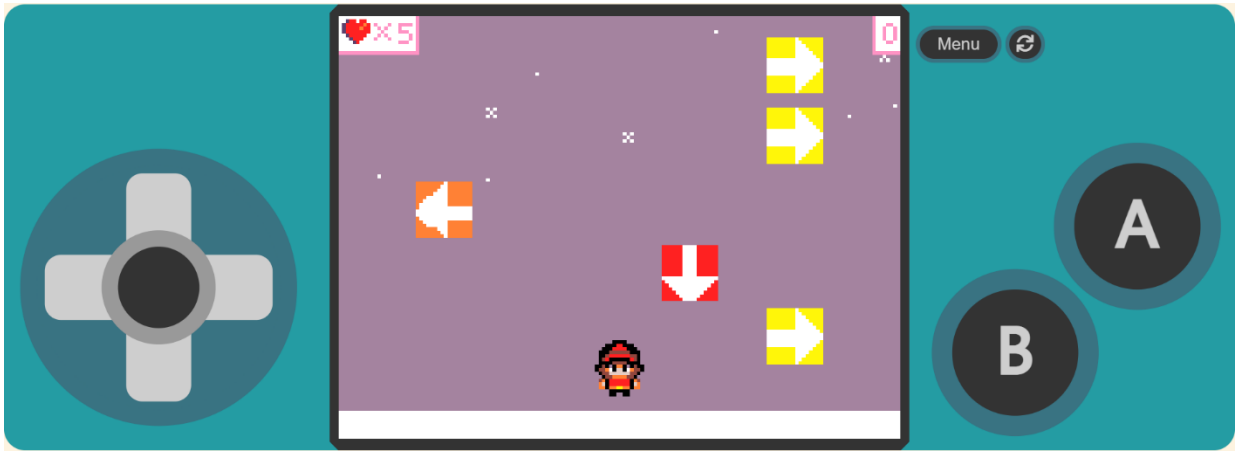
FIRST LOGIC TASK



Description

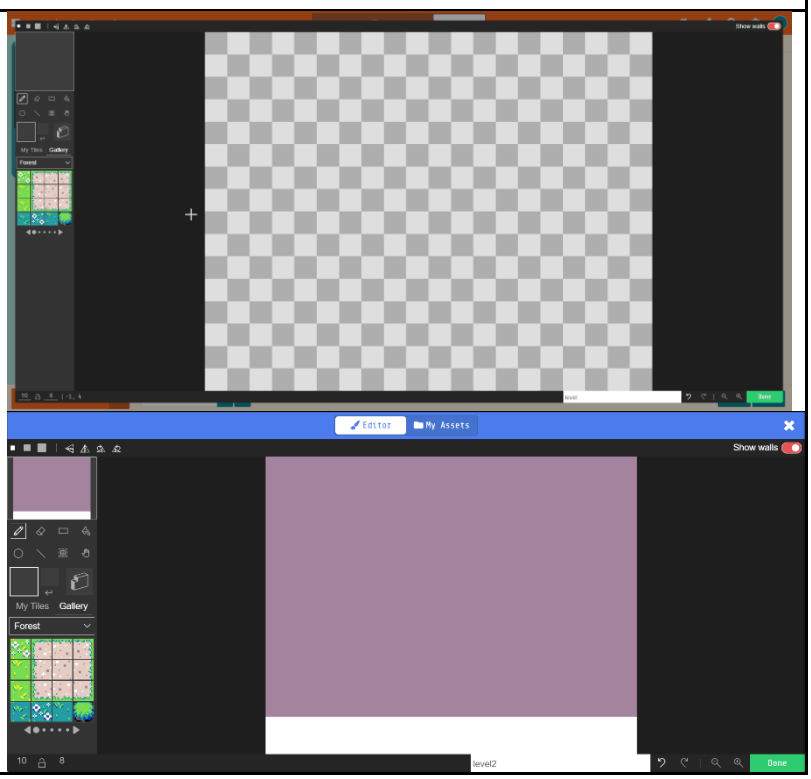
In this exercise, we will create a "Guitar Hero" style video game, where the elements will appear from the top of the screen and our player will have a designated position for each key.

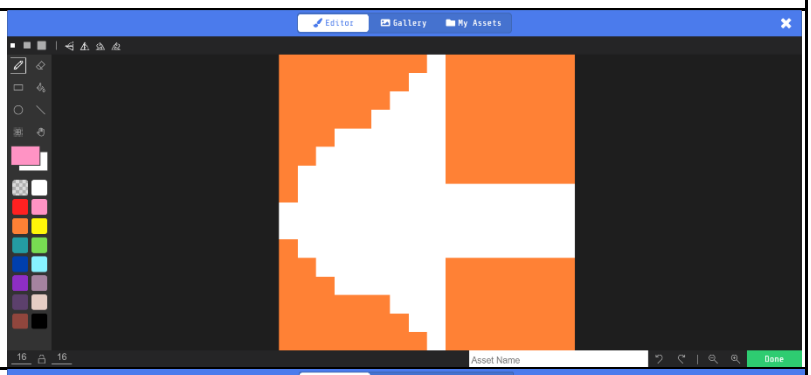
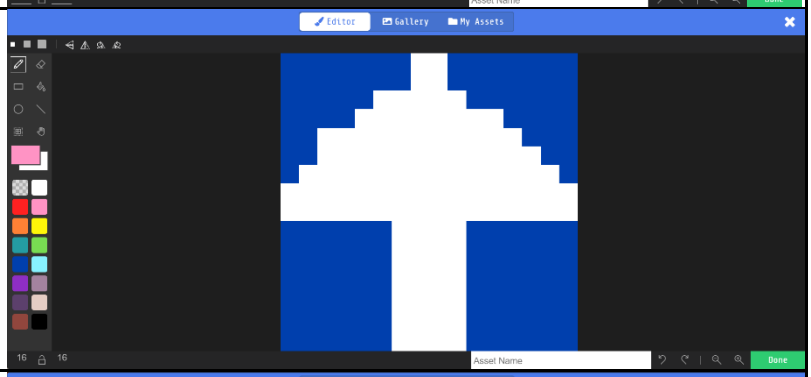
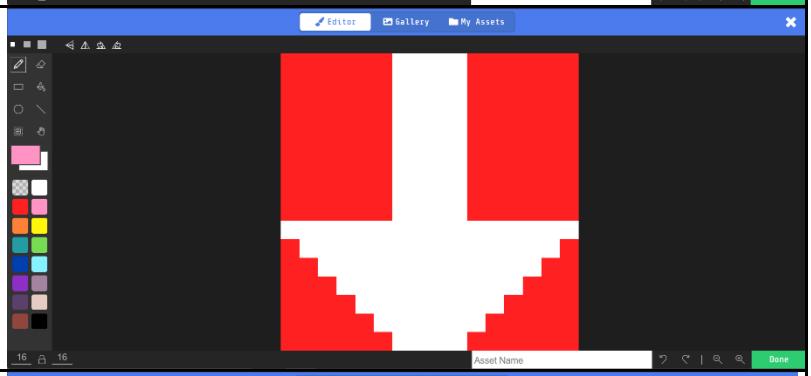
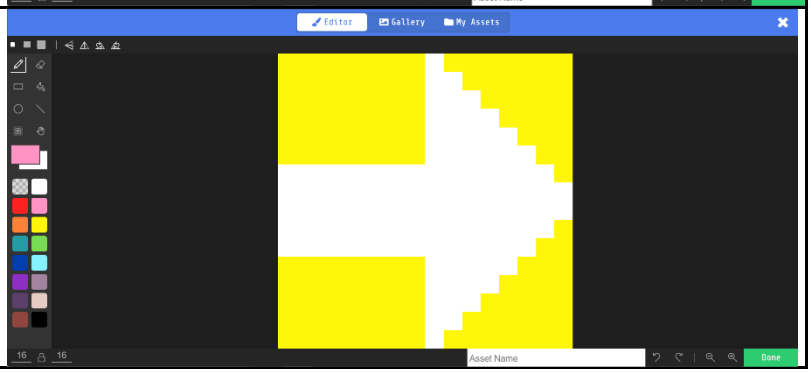
To do that we go to **MakeCode Arcade** and we realise the following operation.

Goals

- Work with game logic in MakeCode Arcade.
- Work with and understanding variables in MakeCode Arcade.
- Assign a player position to each key.
- Explore melodies and sounds in the program and their applications.
- Increase the difficulty by changing the speed.

Game programming

ASSETS CREATION	
MAIN SPRITE CREATION	
We recommend using a 16x16 px grid for Steve's sprite.	
TILEMAP CREATION	
We recommend using a 10x8 px matrix.	

ADITIONAL SPRITE CREATION	
We recommend using a 16x16 grid for the left Sprite	
We recommend you using a 16x16 grid for the up sprite	
We recommend you using a 16x16 grid for the down Sprite	
We recommend you using a 16x16 grid for the right sprite.	

MAIN PROGRAMMING

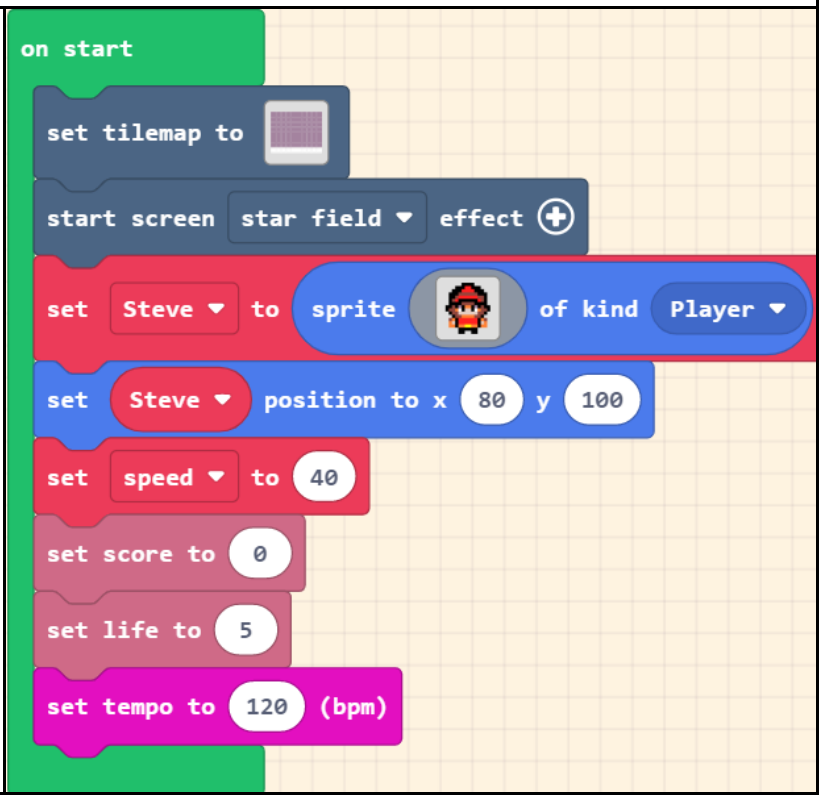
ON START GAME CREATION

The first step is to create an "on start" block and create our tile map. Additionally, we can add an effect to start the game.

The next step is to create the sprite for our player. We choose an image from the gallery and apply it. Then, we set a default position for the character.

We add the speed of the notes and set it to 40. We also add a score counter and set it to 0, and we give the player 5 lives.

Lastly, we set the tempo for the music playback.



NOTES SPAWN MECHANIC

What we will do now is create the game mechanics where the game randomly chooses one of the 4 options every half a second, set an initial position for each arrow, and increase the spawn speed. Using the "on game update every" block, we will make the actions inside it happen every half a second. With a variable called "lane," we will set a lane for each arrow to descend, and we will make it choose one of the four lanes randomly.

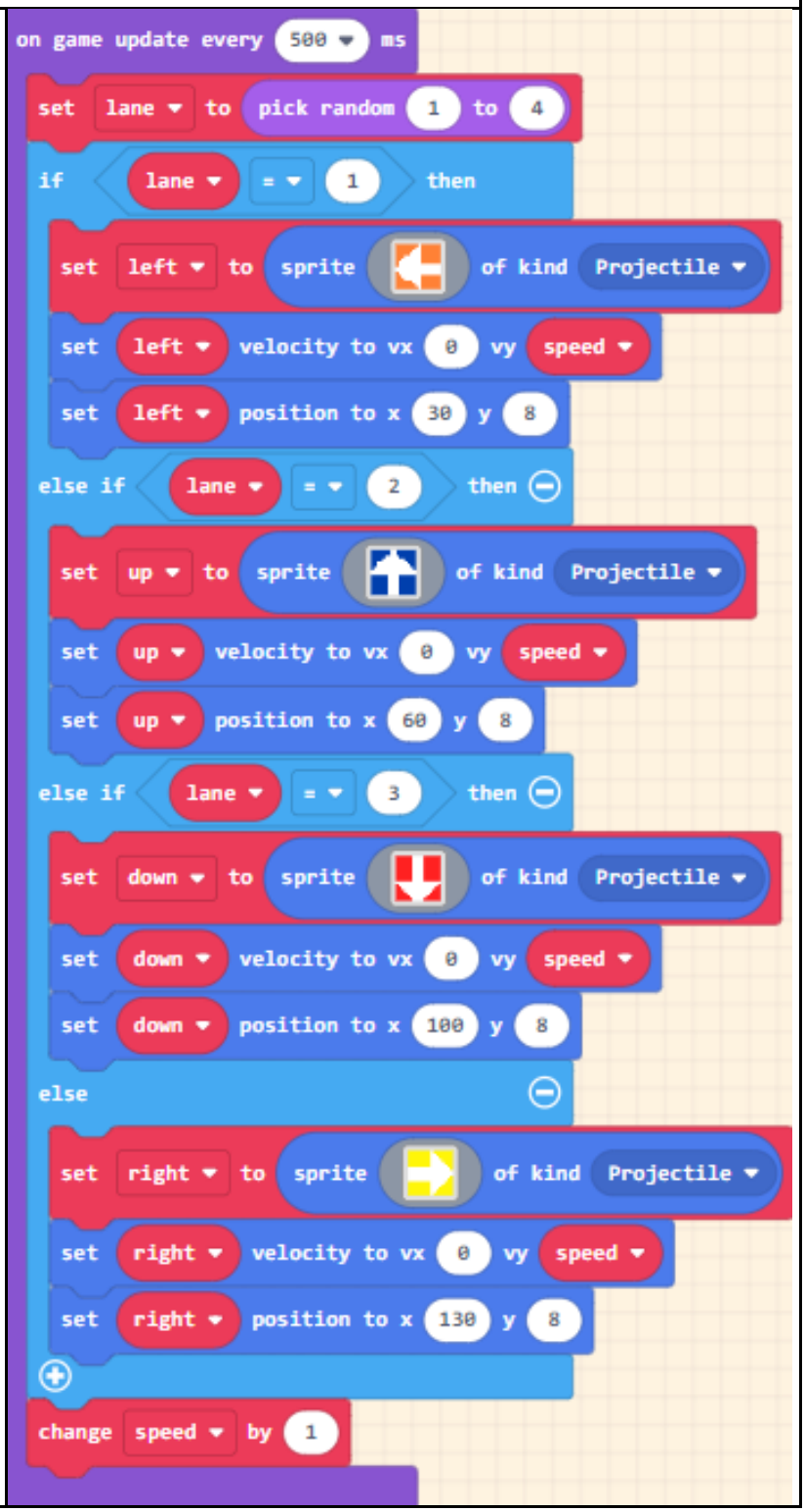
Now, using an "if" statement, we will give instructions for lane number 1. In this lane, a left-facing arrow sprite will appear as a projectile type. We set its x velocity to 0 and add the "speed" variable to its y velocity. Finally, we set the initial position for the left arrow, which will be x = 30 and y = 8.

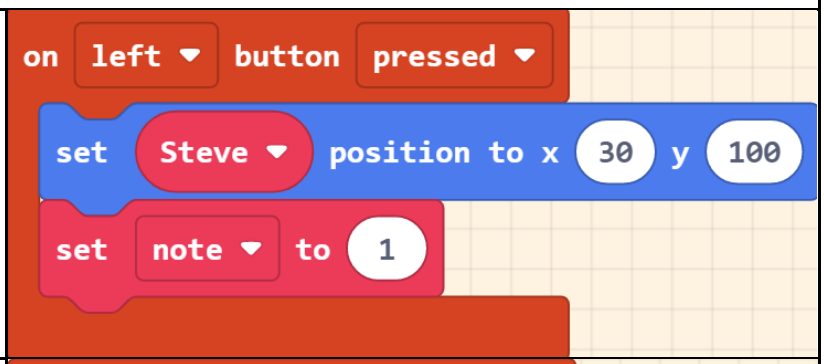
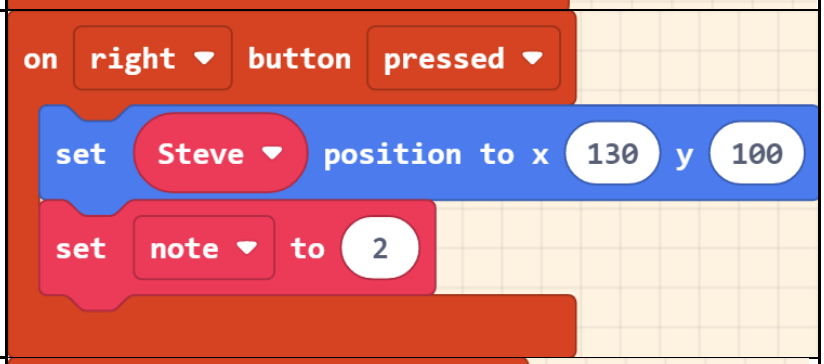
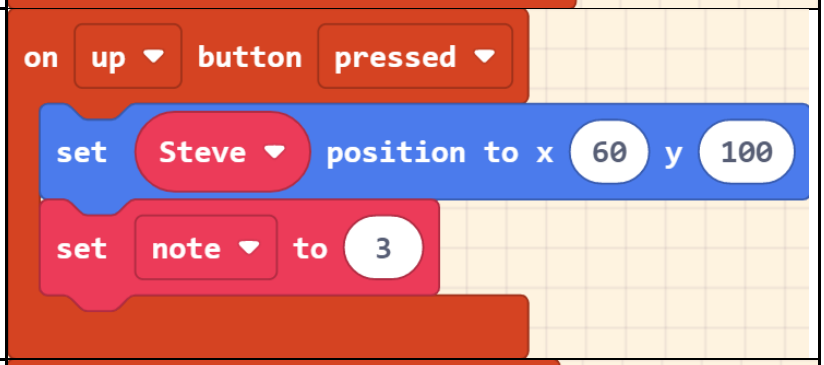
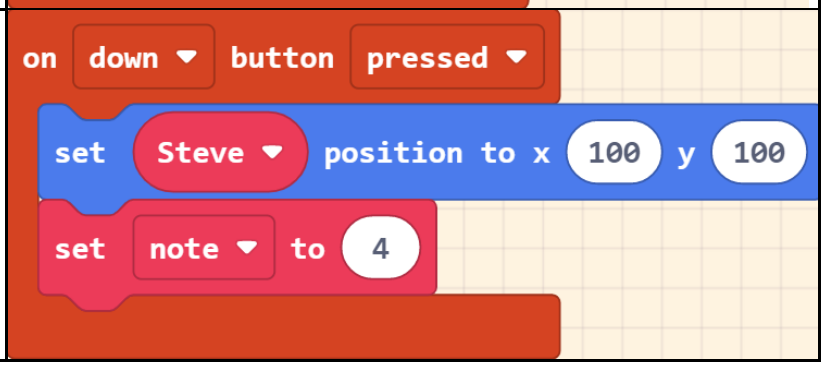
In line 2, we will use the up-facing arrow sprite and give it the same velocity parameters, but we change the position to x = 60 and y = 8.

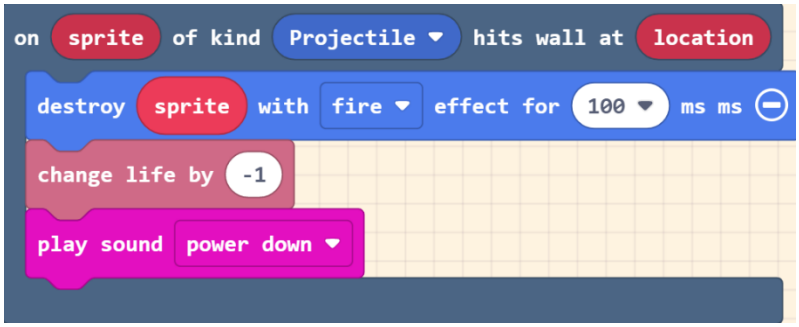
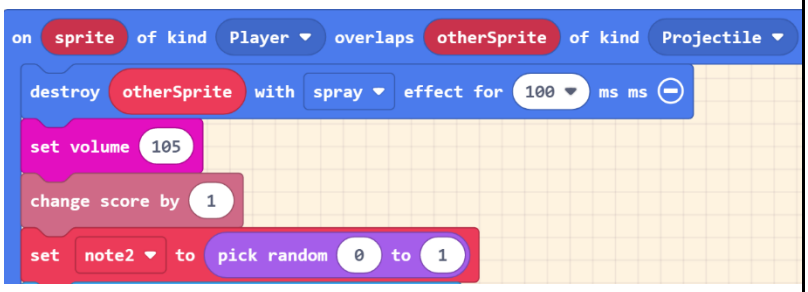
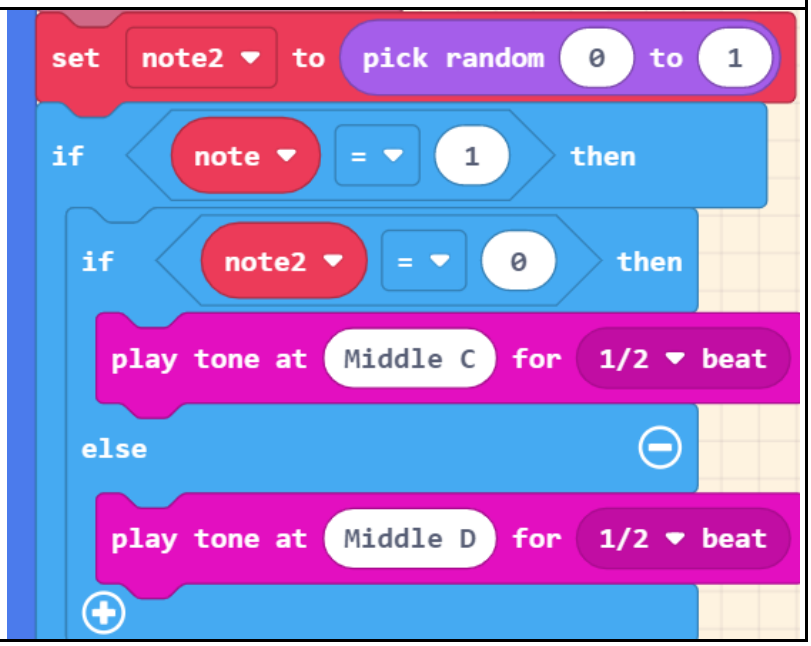
For the characteristics of line 3, we will use a down-facing arrow. We set the same velocity parameters but change the position to x = 100 and y = 8.

For the last lane, we will use a right-facing arrow with the same velocity parameters as the previous arrows, but we change the position to x = 130 and y = 8.

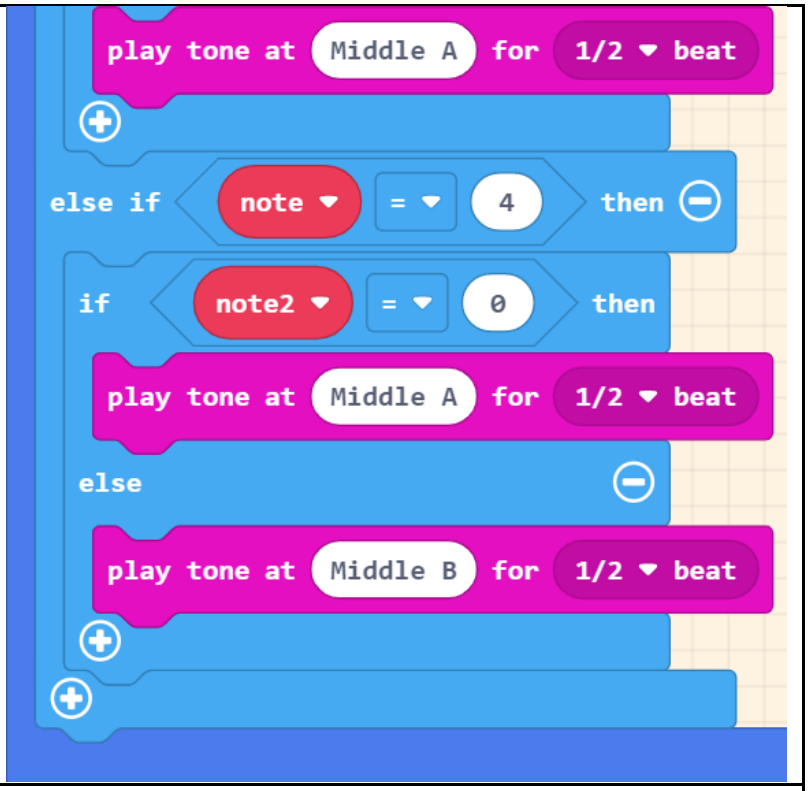
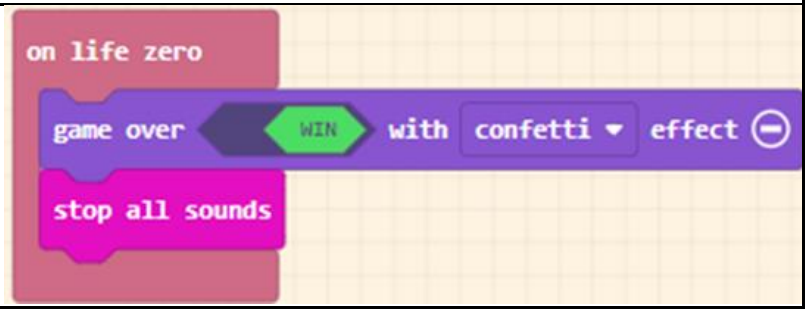
Lastly, if we want to add difficulty, we can add the block "set speed by 1". This way, the speed of arrow spawn will increase by one.



PLAYER MOVEMENT MECHANIC	
Now it's time to give movement to the player so that they can catch the falling arrows. To do this, we will set the player's x position to be the same as the left arrow's x position when the left button is pressed, and we set the note to 1.	
When the right button is pressed, we will set the player's x position to be the same as the right arrow's x position, and we will set the note to 2.	
When the up button is pressed, we will set the player's x position to be the same as the up arrow's x position, and we will set the note to 3.	
When the down button is pressed, we will set the player's x position to be the same as the down arrow's x position, and we will set the note to 4	

LOSING LIFE MECHANIC	
Now, in order to prevent the uncollected arrows from accumulating at the bottom, we will add the "on sprite of kind 'projectile' hits wall" block. With this, we will make the arrows destroy themselves. We can decorate it with a fire effect and add a sound. Additionally, the player will lose a life each time he fails to collect an arrow.	
NOTES AND SCORE MECHANIC CREATION	
Now, we will create an "on sprite of kind 'player' overlaps 'projectile'" block to destroy the arrow when the player collects it, along with a special effect. We will also increase the score by 1 and randomly assign two different notes with a volume of 105 for the player to collect.	
To set the parameters for Note 1 (left arrow), we will use an "if" statement to check if the note equals 1. Inside this "if" statement, we will use an "if-else" statement to handle two different sounds using the variable "note2". This way, it will randomly choose one of the two sounds.	

Now it's time to work on Note 2 (right arrow). We will follow a similar process as before by nesting the conditions. Inside an "if" statement, we will add another "if-else" statement with the condition for Note 2 and two different sounds. This way, when we catch this arrow, it will play one of the two sounds randomly.	
The same process applies to Note 3 (up arrow). We add another nested condition within the existing ones, setting the condition note = 3. Then, we introduce two different tones for the up arrow, so it will randomly choose one of them when caught.	

Finally, for Note 4 (down arrow), we follow the same process. Within the nested condition, we set the condition note = 4. Then, as before, we add two tones for the down arrow, so it will play one of them randomly when caught.	
GAME OVER MECHANIC CREATION	
The last step is to add an "on life zero" block and add effects inside it so that when our life reaches zero, a game over message appears and the sounds stop.	

With this programming, our player will appear on the screen and will have to move sideways to collect the falling arrows and create a melody while scoring points. When the player misses a note, they will lose one life, and if their life reaches zero, the game will end.

Now, it is your turn to personalize and add content to it. Here is our version for inspiration:

https://makecode.com/_069M2R7W0dX8

Glossary

Conditionals: Sequence of instructions that are executed based on the value of a condition.

Example: If, If...Else

If: Conditional statement that, based on the result of a logical operation, executes a sequence of instructions or skips them.

If-Else: Conditional statement that, if a condition is met, executes one sequence of instructions; otherwise, it executes a different sequence.

If...Else if: Sequence of conditionals in which we pass, in an orderly manner, from one condition to another until one of them is met.

Comparison Operators: Operators that compare one value to another and are used within a condition.

Variables: A space associated with an identifier that holds a value, which can be modified.

Functions: A subprogram that contains a set of instructions and can be executed from the main program by calling it.

Sequences: A programmed action that the computer performs in order.

Event: Executes a sequence of instructions when an external event occurs.

Player: A participant in a game.

Acceleration: The change in velocity per unit of time.

Velocity: A physical quantity that relates position to the change in time.

Scene: The space where the video game takes place.

Randomness: The generation of numbers with equal probability.

Score: The total points a player obtains by performing certain interactions.

Life: A resource that the player has to continue playing. Once all lives are lost, the game is over.

Game Over: The game has ended. It usually displays scores and asks if you want to play again.

Music: A combination of sounds and silences that compose a rhythm.

Game Genre: A classification of video games based on their gameplay.

Effect: Something applied to the scene, object, character, or other elements to convey realism or a sensation within the game.

Colour Palette: A panel with a variety of colours that allows selecting a colour to apply to elements in the video game.