

**Discovering Knowledge**

**COURSE: CEL 220**

**COMPUTER ARCHITECTURE & LOGIC DESIGN**

**PROJECT REPORT**

**CLASS: BSE – 3B (FALL - 2023)**

**Flappy Bird**

**Group Members**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Name** | **Enrollment #** |
| 01 | RAJA MUHAMMAD HAMMAD (Team lead) | 02-131222-039 |
| 02 | ABDUL AHAD KHAN | 02-131222-101 |
| 03 | ABDULLAH | 02-131222-099 |

**Submitted to:**

Course Instructor: Engr. Ramsha Mashood

Lab Instructor: Engr. Noor us Sabah

**Department of Software Engineering**

**BAHRIA UNIVERSITY KARACHI CAMPUS**

**ABSTRACT**

**Flappy Bird Game: A Keyboard-Controlled Arcade-Style Adventure**

This project centers around the development and implementation of a digital recreation of the iconic Flappy Bird arcade-style game. The game involves controlling a bird that perpetually moves towards the right, requiring the player to navigate it through randomly positioned pairs of pipes, each featuring identically sized gaps at varying heights. The gameplay mechanic involves the bird's automatic descent, with the sole means of ascending being the player's interaction via tapping the 'f' button on the keyboard. The project involves the utilization of programming languages and game development tools to create an engaging and challenging gaming experience, replicating the simple yet addictive nature of the original Flappy Bird game. Through the design and execution of this project, it aims to offer entertainment, engage users in skill-based gameplay, and showcase the technical aspects of game development and user interaction.

TABLE OF Contents

[1. INTRODUCTION 1](#_Toc155564629)

[2. PROBLEM STATEMENT 1](#_Toc155564630)

[3. PROPOSED SOLUTION 1](#_Toc155564631)

[3.1. FEATURES OF THE PROJECT 1](#_Toc155564632)

[3.2. METHODOLOGY 2](#_Toc155564633)

[Game Logic Design: 2](#_Toc155564634)

[Bitmap Graphics Representation: 2](#_Toc155564635)

[User Input Handling: 2](#_Toc155564636)

[Collision Detection: 2](#_Toc155564637)

[Game Loop and Flow: 2](#_Toc155564638)

[3.3. TECHNOLOGIES 2](#_Toc155564639)

[4. PROJECT SCOPE 3](#_Toc155564640)

[5. MODULE DISTRIBUTION 3](#_Toc155564641)

[6. CODE 3](#_Toc155564642)

[7. INTERFACES 13](#_Toc155564643)

[8. CONCLUSION 14](#_Toc155564644)

[9. REFERENCES 14](#_Toc155564645)

# INTRODUCTION

Flappy Bird is an arcade-style game in which the player controls the bird, which moves persistently to the right. The player is tasked with navigating the bird through pairs of pipes that have equally sized gaps placed at random heights. It automatically descends and only ascends when the player taps the ‘f’ button on the keyboard.

# PROBLEM STATEMENT

Creating a version of the Flappy Bird game in MIPS assembly language with bitmap graphics poses a unique set of challenges. The game involves controlling a bird that moves continuously to the right, navigating through pairs of pipes with randomly placed gaps. The bird automatically descends and ascends only when the player taps the 'f' button on the keyboard. The challenge lies in implementing game logic, user input handling, and bitmap graphics representation within the constraints of the MIPS architecture.

# PROPOSED SOLUTION

The proposed solution is to develop a Flappy Bird game in MIPS assembly language using bitmap graphics. This involves designing the game's architecture, implementing game logic for bird movement and pipe generation, handling user input, and integrating bitmap graphics for visual representation. The solution aims to provide an engaging and functional version of Flappy Bird while adhering to the limitations of the MIPS architecture and bitmap graphics.

## FEATURES OF THE PROJECT

In our game, the image moving is the most important function we need to implement. Besides the background horizontal moving, the vertical jumping combining keyboard input is our second challenge. The bird briefly flaps upward each time the player taps the ‘f’ button; if the button is not tapped, the bird falls due to gravity.

## METHODOLOGY

## Game Logic Design:

Define the game rules, including bird movement, pipe generation, and user input handling.

Implementation for continuous rightward movement, automatic descent, and ascension upon the ‘f' button tap.

## Bitmap Graphics Representation:

Explore MIPS assembly language capabilities for handling bitmap graphics.

Design and create bitmap images for the bird, pipes, and other relevant game elements.

Develop functions to load, display, and update bitmap graphics on the screen.

## User Input Handling:

Establish mechanisms for detecting 'f' button taps and responding to user input.

Ensure accurate and responsive control of the bird's ascension based on user actions.

## Collision Detection:

Implement collision detection to identify interactions between the bird and pipes. Define outcomes of collisions, such as game over scenarios.

## Game Loop and Flow:

Construct a game loop to iterate through key steps in continuous gameplay. Manage the flow of the game, including start, pause, and restart functionalities.

## TECHNOLOGIES

Mars Simulator was employed in our project. Our project was created using the MIPS Assembly language bitmap.



# PROJECT SCOPE

It's an easy game with endless levels. This game is playable by players of nearly age above 5 years. They can have some enjoyment from it throughout their free time. Additionally, it is a simple game that doesn't require complicated instructions to play. One can play this game for hours on end because it is so addictive. This game can be played on an Android device.

# MODULE DISTRIBUTION

* ***RAJA MUHAMMAD HAMMAD***

Implementation of pipe, coding of top refresh and bot refresh, Coding of bird, using loop.

* ***ABDUL AHAD KHAN***

Designing of bitmap display, coding of reset of game and exit screen.

* ***ABDULLAH***

Coding of jump Method in the game

# CODE

# Bitmap Display Configuration:

# - Unit width in pixels: 8

# - Unit height in pixels: 8

# - Display width in pixels: 256

# - Display height in pixels: 256

# - Base Address for Display: 0x10008000 ($gp)

#

#

#####################################################################

.data

displayAddress: .word 0x10008000

r: .word 4

green: .word 0x00ff44

blue: .word 0x00a2ff

yellow: .word 0xfbff00

firstval: .word 0xffff0000

secondval: .word 0xffff0004

asciif: .word 102

time: .word 250

.text

main:

lw $t0, displayAddress # $t0 stores the base address for display

lw $t1, blue # $t1 stores the yellow and blue colour code

lw $t2, green # $t2 stores the green colour code

lw $t3, time # $t3 sotees the time that syscall waits

move $t4, $zero # $t4 stores the counter for the units

li $t6, 1792 # $t6 stores the address for the corner of the bird

li $t7, 1020 # $t7 stores the edited address for the corner of the pipe

lw $s0, firstval # s0 stores address of the first val - whether there is input

lw $s1, secondval # s1 stores address of the second val - input

lw $s2, asciif # s2 stores ascii code for f

li $t8, 1024 # t8 stores the address for the corner of the pipe

li $t9, 8 # t9 stores the pipe offset

li $s7, 0 # stores the randomized value

li $s6, 128 # stores the length of the display

li $a3, 4 # stores a value to check in ending

#for score time

li $v0,30 # get start timestamp in a0:a1

syscall

#save a0:a1 somewhere

while:

add $t5, $t4, $t0 # t5 gets updated with the next unit

sw $t1, 0($t5) # paint the first unit on the second row blue. Why +128?

add $t4, $t4, 4 # increment to next unit

bne $t4, 4096, while # jump back if the screen is not fully painted

Bird:

lw $t1, yellow

add $t6, $t6, 28

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

lw $s5, 0($t5)

sw $t1, 0($t5) # paint the bird

beq $s5, $t2, Exit

add $t6, $t6, 8

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

lw $s5, 0($t5)

sw $t1, 0($t5) # paint the bird

beq $s5, $t2, Exit

add $t6, $t6, 120

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

sw $t1, 0($t5) # paint the bird

add $t6, $t6, 4

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

sw $t1, 0($t5) # paint the bird

add $t6, $t6, 4

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

sw $t1, 0($t5) # paint the bird

add $t6, $t6, 4

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

lw $s5, 0($t5)

sw $t1, 0($t5) # paint the bird

beq $s5, $t2, Exit

add $t6, $t6, 116

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

lw $s5, 0($t5)

sw $t1, 0($t5) # paint the bird

beq $s5, $t2, Exit

add $t6, $t6, 8

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

lw $s5, 0($t5)

sw $t1, 0($t5) # paint the bird

beq $s5, $t2, Exit

sub $t6, $t6, 292

j Loop

Loop:

jal Pipe

lw $t1, blue

lw $s3, 0($s0)

bne $s3, $zero, Jump

add $t6, $t6, 28

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

sw $t1, 0($t5) # paint the bird

add $t6, $t6, 8

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

sw $t1, 0($t5) # paint the bird

add $t6, $t6, 124

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

sw $t1, 0($t5) # paint the bird

add $t6, $t6, 8

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

sw $t1, 0($t5) # paint the bird

sub $t6, $t6, 40

bge $t6, 3900, Exit

j Bird

Jump:

lw $s3, 0($s1)

bne $s3, $s2, Loop

beq $t6, $zero, Loop

li $a0, 25 # set sleep to 100

li $v0, 32 # set syscall to sleep

syscall # sleep 100

lw $t1, blue

add $t6, $t6, 156

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

sw $t1, 0($t5) # paint the bird

add $t6, $t6, 4

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

sw $t1, 0($t5) # paint the bird

add $t6, $t6, 4

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

sw $t1, 0($t5) # paint the bird

add $t6, $t6, 4

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

sw $t1, 0($t5) # paint the bird

add $t6, $t6, 116

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

sw $t1, 0($t5) # paint the bird

add $t6, $t6, 8

add $t5, $t6, $t0 # t5 gets updated with the corner of the bird

sw $t1, 0($t5) # paint the bird

sub $t6, $t6, 548

j Bird

Pipe:

move $a0, $t3 # set sleep to 100

li $v0, 32 # set syscall to sleep

syscall # sleep 100

sub $t7, $t8, $t9

add $t7, $t7, $s7

lw $t1, blue

add $t7 $t7, 8

add $t7 $t7, 512

TopRefresh:

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint out the pipe

sub $t7, $t7, 128

bge $t7, $zero, TopRefresh

add $t7, $t7, 2048

BotRefresh:

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint out the pipe

add $t7, $t7, 128

ble $t7, 4096, BotRefresh

sub $t7, $t8, $t9

add $t7, $t7, $s7

j TopPipe

Clear:

li $t7, 128

Clear1:

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint out the pipe

add $t7, $t7, 128

bne $t7, 4096, Clear1

li $t7, 124

Clear2:

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint out the pipe

add $t7, $t7, 128

bne $t7, 4092, Clear2

li $t7, 100

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t2, 0($t5) # paint the pipe

sub $t7, $t7, 4

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t2, 0($t5) # paint the pipe

sub $t7, $t7, 4

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t2, 0($t5) # paint the pipe

sub $t7, $t7, 120

add $t7, $t7, 2040

add $t7, $t7, $s7

j BotPipe

TopPipe:

beq $t7, 100, Clear

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t2, 0($t5) # paint the pipe

sub $t7, $t7, 4

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t2, 0($t5) # paint the pipe

sub $t7, $t7, 4

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t2, 0($t5) # paint the pipe

sub $t7, $t7, 120

bge $t7, 4, TopPipe

add $t7, $t7, 2040

add $t7, $t7, $s7

BotPipe:

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t2, 0($t5) # paint the bird

add $t7, $t7, 4

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t2, 0($t5) # paint the bird

add $t7, $t7, 4

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t2, 0($t5) # paint the bird

add $t7, $t7, 120

ble $t7, 4094, BotPipe

add $t9, $t9, 4

bge $t9, 132, Reset

jr $ra

Reset:

beq $t3, $zero, continue

sub $t3, $t3, 25

continue:

li $a1, 12 # set max to 8

li $v0, 42 # set syscall to random int

syscall # get random int

li $t7,3968

reset1:

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint the bird

sub $t7, $t7, 128

bge $t7, $zero, reset1

li $t7, 3972

reset2:

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint the bird

sub $t7, $t7, 128

bge $t7, 4, reset2

li $t7, 3976

reset3:

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint the bird

sub $t7, $t7, 128

bge $t7, 8, reset3

mult $a0 $s6

mflo $s7

sub $t7, $t7, $s7

li $t9, 0

jr $ra

Exit:

lw $t1, blue

move $t4, $zero # $t4 stores the counter for the units

li $v0, 30

syscall

End:

add $t5, $t4, $t0 # t5 gets updated with the next unit

sw $t1, 0($t5) # paint the first unit on the second row blue. Why +128?

add $t4, $t4, 4 # increment to next unit

bne $t4, 4096, End # jump back if the screen is not fully painted

li $t7, 1960

lw $t1, yellow

G:

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint the bird

add $t7, $t7, 4

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint the bird

add $t7, $t7, 4

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint the bird

add $t7, $t7, 4

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint the bird

add $t7, $t7, 116

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint the bird

add $t7, $t7, 128

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint the bird

add $t7, $t7, 8

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint the bird

add $t7, $t7, 4

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint the bird

add $t7, $t7, 116

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint the bird

add $t7, $t7, 12

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint the bird

add $t7, $t7, 116

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint the bird

add $t7, $t7, 4

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint the bird

add $t7, $t7, 4

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint the bird

add $t7, $t7, 4

add $t5, $t7, $t0 # t5 gets updated with a piece of the pipe

sw $t1, 0($t5) # paint the bird

beq $a3, $zero, done

move $a3, $zero

li $t7, 1984

j G

DisplayScore:

# Store the base address for display in a temporary register

move $t0, $gp

# Load the time value into $t5 (assuming time is stored in $t5)

lw $t5, time # Load the time value from the memory location

# Calculate the memory address where the score will be displayed

# Update memory\_location with the specific memory address to display the score

li $t6, 10 # Divider for extracting individual digits

li $t7, 4 # Counter for four digits (adjust as per your time format)

li $t8, 48 # ASCII value for '0'

# Set up memory location where the score will be displayed

li $t9, 40 # Update memory\_location with the specific memory address

DisplayLoop:

bgtz $t7, ExitDisplay # Exit loop if counter is 0

div $t5, $t6 # Divide time by 10 to get the digit at the current position

mfhi $t1 # Remainder has the digit to be converted to ASCII

add $t1, $t1, $t8 # Add '0' to convert to ASCII

sw $t1, ($t9) # Store the ASCII character in memory

mflo $t5 # Update $t5 with quotient for next iteration

addi $t9, $t9, 4 # Move to the next memory location

sub $t7, $t7, 1 # Decrement counter

j DisplayLoop # Jump back to DisplayLoop

ExitDisplay:

jr $ra # Return from the subroutine

done:

li $v0, 10 # terminate the program gracefully

syscall

# A screenshot of a computer Description automatically generatedINTERFACES

A screenshot of a computer

Description automatically generated

# CONCLUSION

Flappy bird is a simple game of the infinite level type. People of almost every age can play this game. It can be played in their leisure time for entertainment. It is also an easy-to-play game which does not require any complex procedure to play it. It is an addictive game, and a person can play it for hours. It is possible to play this game on android.

# REFERENCES

Patterson, D. A., & Hennessy, J. L. (2017). Computer Organization and Design: The Hardware/Software Interface (5th ed.). Morgan Kaufmann.

MIPS Assembly Language Programming. Link For Reference

MIPS Assembly Language Programming which we learn in the Lab.