Anteater Project

Gabriela Gonzalez (gng2112) & Chirag Chaturvedi (cc4880)

Contents

1. Introduction
2. Game Logic
3. Software Design
4. Hardware Design
5. Algorithms
6. Memory Budget
7. Goals & Milestones

1. Introduction

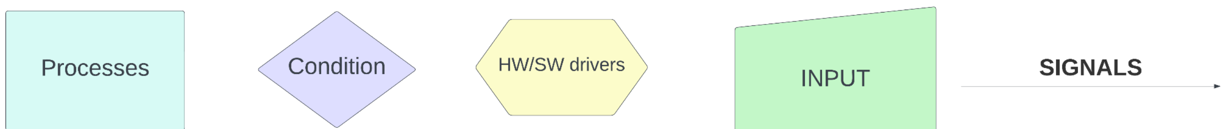
We plan on recreating Anteater, the arcade game released by Tango Electronics in 1982[1] where players must use a joystick to steer the anteater's tip of its tongue through an underground maze to collect ant larvae, ants, worms, and queen ants. Players also must utilize a button to retract the tongue to prevent losing a life in situations such as having an ant hit the tongue or hitting a worm anywhere besides from behind. Players can advance levels by collecting all ant larvae present in the maze. If a queen ant is collected, then all creatures go away for a short period of time to facilitate larvae collection. The objective of the game is to obtain the highest score possible while collecting as many creatures as possible.

1. https://en.wikipedia.org/wiki/Anteater_(video_game)

2. Game Logic

Data Flow Diagram of AntEater Game

Input from Controller
libusb Driver
Tongue tip Position Calculation
Rest of the tongue position Calculation
Check if it hits a wall
Yes
It stops moving
No
Chcek if the tip hit a queen ant
Yes
Remove all ants and worms for 3 seconds
+1000
If no. of Larvae = 0
Yes
Go to next Level
Check if the tips hit a larvae
Yes
+200
No
Check if the tip hits an ant
Yes
+100
No
Score Calculator
Check if the tips hit a Worm
Yes
Check if Back or front of the worm
Back
+10
No
Front
Check if an ant hits the tongue
Yes
Reduce life by 1
No
Update Position of the tip
Video Driver
Update rest of the tongue position
Check if lives equal 0
Yes
End Game
Sound Driver



Legend

Scoring Logic

Creature	Score Addition
Larvae	+10
Ant	+100
Worms	+200
Queen Ant	+1000

Process Details

Input from the controller: Users can control the horizontal and vertical position of the tip of the tongue with the help of the joystick on the controller. They can also retract the tongue with the help of a button on the controller. We will be using an Xbox 360 controller which connects to the PC with a USB connector, this enables us to use the library “libusb” for getting signals from the controller.

Tongue Tip Position Calculation: After the input has been received, the position of the tip will be calculated, this is going to be pretty simple as we don’t have to consider how much the joystick has been pushed and all we have to worry about is the angle as the tongue will only be moving either vertically or horizontally. During this process, we will be sending the updated position continuously to the video driver. If the next position is the wall and not a free path, the tongue will stop moving and it will wait for the next input.

Tongue Position Calculation: The tip of the tongue and the tongue are 2 separate entities in the game as they interact with other objects in the game very differently from each other, as the tip moves it leaves a trail of the tongue behind it, and we will have to keep track of all the positions it is on as collision of the tongue with ants or worms can lead to different outcomes, the user needs to keep track of the tongue as well. During this process as well we will have to constantly send position data to the video driver so that it can be reflected on the screen.

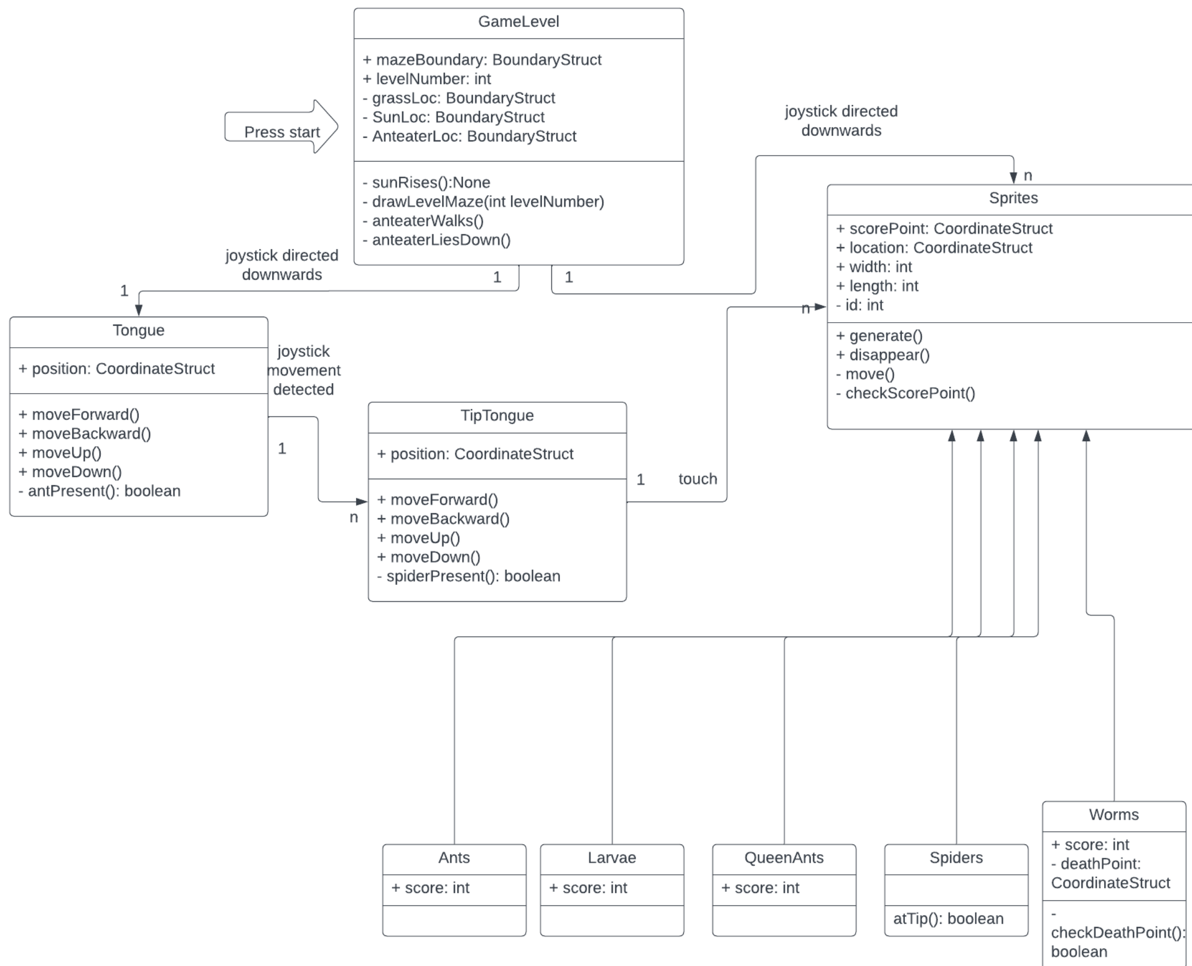
Collision of the tip with Ants, Worms, and Larvae: If during movement the tip collides with an Ant then the Ant disappears and we add 100 points to the score, this data is immediately sent to the video driver, if it collides with a worm then we check whether if it has collided with the front of the worm or the back If its the front, then we reduce the lives by 1 and this data is sent to the video driver to show on the screen, else if the tongue hits the back of the worm then the worm disappears and we add 200 points to the score, this data is immediately sent to the video driver

and is reflected on the screen. If the tongue hits larvae, the larvae are made to disappear and a score of +10 is added, if it is the last larvae present in the game, then we progress to the next level, this data is also sent to the video driver immediately.

Collision of the tongue with Ants and Worms: If during movement an Ant collides with the tongue, then the number of lives is reduced by 1, if a worm collides with a tongue then it just passes through, this data is immediately sent to the video driver so that It can be reflected on the screen, along with that we also check if the no. of lives = 0, because if it does then the game ends there and this data is also immediately sent to to the video driver.

3. Software Design

UML Diagram for Anteater Game



In this section, we will discuss the object-oriented design decisions we anticipate implementing. The UML diagram above illustrates the current plan for implementation of Anteater.

The GameLevel class will create the scaffolding for the level, like setting the maze boundaries, that is contingent on the degree of difficulty for the level. Additionally, grass will be placed on top of the maze to emulate the original game graphics. For the maze and grass, there will be a BoundaryStruct created to indicate the exact position of the negative space in the maze.

Once the maze has been set, the anteater can walk out and place itself at the opening to prepare for gameplay. In the midst of this, the sun will rise or set depending on the level count. Depending on the level, the anteater will be standing on its legs or lying down, so we added a function to take this into account.

When the joystick is moved, the tongue will move and its position will be stored so that we can display the path the tongue has taken. Additionally, the tongue position will determine some end-game conditions that will be specified in the Sprites class. The tongue will have functions that will be triggered by the joystick to move forward, backward, up, and down. The Tongue class will also have methods such as antPresent(), an endgame condition.

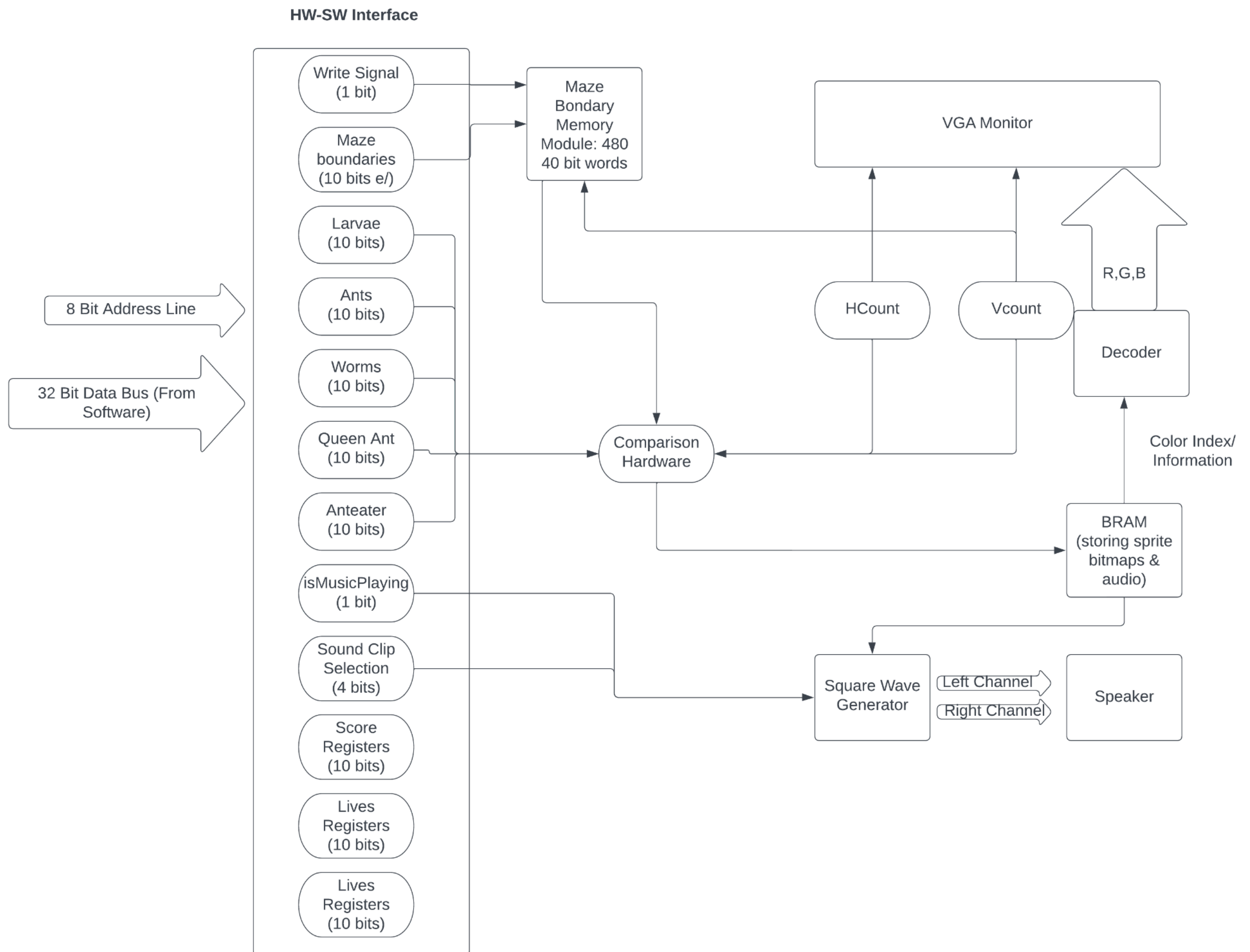
Following the tongue will be the TipTongue to denote the attributes for the tip of the anteater tongue. Only the tip of the tongue can touch the sprites to acquire points. It will have the same attributes as the tongue class class. Additionally, spiderPresent() method will be utilized to check if an ant has hit the tongue or if a spider has reached the tip of the tongue, an endgame condition.

The Sprites parent class will hold information about its location, width, and length, id (identifier). The attribute scorePoint corresponds to the spot on the sprite that the tip must touch in order to score a point and cause the sprite to disappear. The functions generate(), disappear(), move() will change the location of the sprite while the checkScorePoint() will check if a sprite has been "eaten."

The Ants, Larvae, QueenAnts, and Worms classes will have an attribute score to indicate how much they are worth. The Spiders class will just check if a spider is at the tip of the tongue.The Worms class will have a score similar to the other sprites, but it will also have a deathPoint attribute. Since a worm can only be eaten from behind, deathPoint indicates the point the tip needs to hit in order to lose. There will be a checkDeathPoint() method to check if the death point has been touched.

4. Hardware Design

Hardware System Design



Register Descriptions

Write Signal:
Negative Maze Boundary Left: 10 bit integer storing left boundary of a maze boundary
Negative Maze Boundary Right: 10 bit integer storing right boundary of a maze boun3dary
Negative Maze Boundary Top: 10 bit integer storing the top boundary of a maze boundary
Negative Maze Boundary Bottom: 10 bit integer storing bottom boundary of a maze boundary

At least 27 of these boundaries will be required depending on the degree of difficulty of the layer.

Shift: 1 bit shift signal

Ant 1 x position: 10 bits storing the x position of Ant 1
Ant 1 y position: 10 bits storing the y position of Ant 1
…
Worm 1 x position: 10 bits storing the x position of Worms 1
Worm 1 y position: 10 bits storing the y position of Worm 1
…
Spider 1 x position: 10 bits storing the x position of Spider 1
Spider 1 y position: 10 bits storing the y position of Spider 1

There will be n worms, spiders, and worms depending on the degree of difficulty of the layer

Sun x position: 10 bits storing the x position of Sun
Sun y position: 10 bits storing the y position of Sun

Queen Ant x position: 10 bits storing the x position of Queen Ant 1
Queen Ant y position: 10 bits storing the y position of Queen Ant 1

Anteater x position: 10 bits storing the x position of Anteater
Anteater y position: 10 bits storing the y position of Anteater

Score Digit 1 x position: 10 bits storing the x position of Score Digit 1
Score Digit 1 y position: 10 bits storing the y position of Score Digit 1

Score Digit 2 x position: 10 bits storing the x position of Score Digit 2
Score Digit 2 y position: 10 bits storing the y position of Score Digit 2

Score Digit 3 x position: 10 bits storing the x position of Score Digit 3
Score Digit 3 y position: 10 bits storing the y position of Score Digit 3

Score Digit 4 x position: 10 bits storing the x position of Score Digit 4
Score Digit 4 y position: 10 bits storing the y position of Score Digit 4

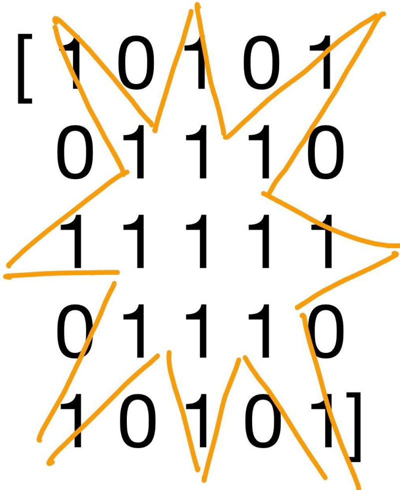
Score Digit 5 x position: 10 bits storing the x position of Score Digit 5
Score Digit 5 y position: 10 bits storing the y position of Score Digit 5

Lives Count x position: 10 bits storing the x position of Lives Count
Lives Count y position: 10 bits storing the y position of Lives Count

5. Algorithms

Sprites Drawing Algorithm

- Having an nxn array of 0's where pixels that will be written will be converted to a 1 to form a shape



- Timing of the animals: with probability function so that the animals are not predictable
    - In our initial implementation we will be generating them randomly with slight probability.
    - If time permits we will try to implement the timing and movement of other animals using RL in-order to make the game more interesting and harder
- Overall have a 2D array that keeps track of what is there is a wall vs empty vs food and where tongue is. With the different threads, create a mutex so that different gameplay is represented. This will help check when there have been interactions with the tongue that may involve end-game conditions. This table gets updated upon the detection of movement from the joystick and method calls to move sprites.
  [[M M M M E T E M M M M]
  [E E E E E E T T T L L L L]]
    - M for maze boundary
    - E for empty
    - T for tongue
    - L for larvae
    - W for worm
    - Q for queen ant
    - S for spider
    - Su for Sun

Libraries to Leverage

https://github.com/paroj/xpad→ interface with XBox controller

6. Memory Budget

Graphics Memory

Category	**Size (pixels)**	**Total Size (bits) = size * 4**
Sun	32 * 32	4096
Anteater	60 * 32	7680
Worms	32 * 32	4096
Spiders	32 * 32	4096
Larvae	20 * 32	2560
Grass	10 * 10	400
Large blocks negative space in maze	TBD(not constant, changes with lvl)	-
Small blocks negative space in maze	TBD(not constant, changes with lvl)	-
Ant	32 * 32	4096
Queen ant	40 * 32	5120
Tongue	TBD(not constant, changes with movement)	-
Tip of Tongue	10 * 20	800
Score Digit	32 * 32	4096 X5 (for 5 digits)
Lives Digit	32 * 32	4096

Total Memory Budget (bits): TBD

Audio Memory

	Ant eaten	**Worm eaten**	**Endgame**
Time (s)	0.5	0.5	0.5
Frequency (KHz)	48	48	48
Memory (bits)	TBD	TBD	TBD

Total Audio Budget: TBD

7. Goals & Milestones

Goals

1. Minimum Viable Product (MVP): have a correctly laid-out maze and stagnant anteater whose tongue can eat stagnant ants and larvae with a controller.
2. Include all sprites with movement
3. Button can retract tongue
4. Music and sound-effects are incorporated into the game
5. Anteater walks to the start location and walks away at conclusion of level
6. Intralevel screen to dictate level statistics
7. The sky and negative space in the maze change colors in between levels
8. Stretch: Incorporate ML to dictate sprite position

Milestones

1. Finish making all the sprites by March 31st
2. Finish all drivers, have sprites moving on the screen, have a moving tongue with retraction by April 12th
3. Finish incorporating audio, color changes between levels, the intralevel screen, and have the anteater moving by April 26th
4. Incorporate ML by May 10th