SE 101

Introduction to Methods of Software Engineering

Fall 2022

Course Project Final Report

Banana Boys

Name	User ID	User Name
Adam Troiani	[redacted]	actroian
Emerson Dang	[redacted]	edang
Hussein Elguindi	[redacted]	helguind
Michael Xu	[redacted]	m4xu
Muneeb Farrukh	[redacted]	m5farruk

Contents

1 Introduction **2**
- 1.1 Interactive . . . 2
- 1.2 Immersive . . . 2
- 1.3 Customizable . . . 2

2 Background Research **2**

3 Implementation **3**
- 3.1 Hardware . . . 3
  - 3.1.1 Host-Machine Dongle . . . 3
  - 3.1.2 Paddle . . . 3
- 3.2 Cross Device Communication . . . 4
  - 3.2.1 ESP-NOW and Serial I/O . . . 4
  - 3.2.2 Data Serialization and Protocol Buffers . . . 4
- 3.3 Computer Vision . . . 5
- 3.4 QuadTree . . . 6
- 3.5 Polygon-Ball Collision Detection . . . 6
- 3.6 Geohashing . . . 6
- 3.7 Ball Guidance with Paddle . . . 7

4 Contribution of Group Members **7**

5 Final Product Evaluation **8**

6 Design Trade-Offs **9**

7 Future Work **9**

1 Introduction

Brick Breaker is a classic, timeless game. The premise is simple: the player moves a digital "paddle" horizontally to rebound a bouncing ball with the goal of "breaking" an arrangement of rectangular bricks above the paddle. For several of our group members, the first time we played Brick Breaker was in kindergarten on our parent's 2010 Blackberry Curve. It was a staple of our childhoods and we wanted to make our own improved version of the game. Our aim was to improve Brick Breaker by making it more interactive, immersive, and customizable.

1.1 Interactive

Traditionally, the game is played with a keyboard interface, or in the case of the Blackberry, an optical trackpad. Both of these input interfaces fail to simulate the natural action of moving a paddle back and forth. As such, we developed a physical motion controller that lets you control the digital paddle by moving it laterally in real 3D space. When you play our game, it is as if you are holding the paddle in your hand.

1.2 Immersive

Fully committing to simulating the act of physically moving the paddle, our controller provides vibrational haptic feedback to mimic the sensation of the ball bouncing off of the paddle.

1.3 Customizable

It is a well known universal fact that when the player has control over map creation, games become infinitely more fun (take, for instance, Geometry Dash and Super Mario Maker). Our version of Brick Breaker allows the user to design their own arrangement of bricks. One way of doing so is by drawing, either physically or digitally. With this method, the shapes inscribed by lines will be transformed into bricks. To accommodate for players who are artistically challenged, we also provide an automated option to transform a coloured image into a series of bricks. The size of the bricks are determined using a standard image compression algorithm that groups regions of similar colour into a single brick.

2 Background Research

There was much research on the computer vision side of the project. After reading many of the tutorials on opencv.org, it was clear that the cornerstone of computer vision is, in fact, linear algebra, something us 1A Software Engineers are all too familiar with. Almost

all of OpenCV's image processing functions rely on converting the image into a matrix of subpixels, which are then manipulated to achieve the function's desired result. Researching the field of computer vision was especially interesting as it not only allowed us to build upon our programming knowledge, but our linear algebra knowledge as well.

Our group came across a highly relevant study conducted by the University of Waterloo's Haptic Experience Lab that explored the effects of vibrotactile embellishments on player experience. Singhal and Schneider discovered that vibrotactile stimulation has a measurable effect on a player's enjoyment of a game [11]. This inspired us to incorporate coin vibration motors into our paddle in order to generate haptic feedback.

3 Implementation

3.1 Hardware

3.1.1 Host-Machine Dongle

The dongle is an ESP8266 that is connected serially through USB to the host machine. Espressif boards, such as the ESP8266 and the ESP32, were strategically chosen for their ESP-NOW wireless connection protocol capability. The dongle acts purely as a forwarder from the paddle to the host machine, to allow for a wireless playing experience. It listens for serial input, which is usually sent from the Brick Breaker game, then wirelessly transmits the data to the paddle. As for the reverse direction, the dongle listen asynchronously for any data sent to it, which is then transmitted through the wired serial connection. The in-depth implementation of the communication is detailed in subsection 3.2.

3.1.2 Paddle

The paddle is comprised of a 3D printed case that houses the ESP32 (3.3V) and Arduino Nano (5V) microcontrollers along with the HC-SR04 (5V) ultrasonic distance sensor and other components outline below. The Arduino Nano is used for its 5V voltage regulator so that both the ESP32 and the HC-SR04 are provided with 5 volts. The ESP32's on board voltage regulator regulates to 3.3V, which is not enough for the distance sensor. In this configuration, the Arduino Nano can be provided with 7V-12V, while providing adequate power to each of the other components. 4 vibration motors are included to provide the user with physical feedback. These motors require 3.3V and are thus connected to the ESP32's digital pins for controllable power output (on or off).

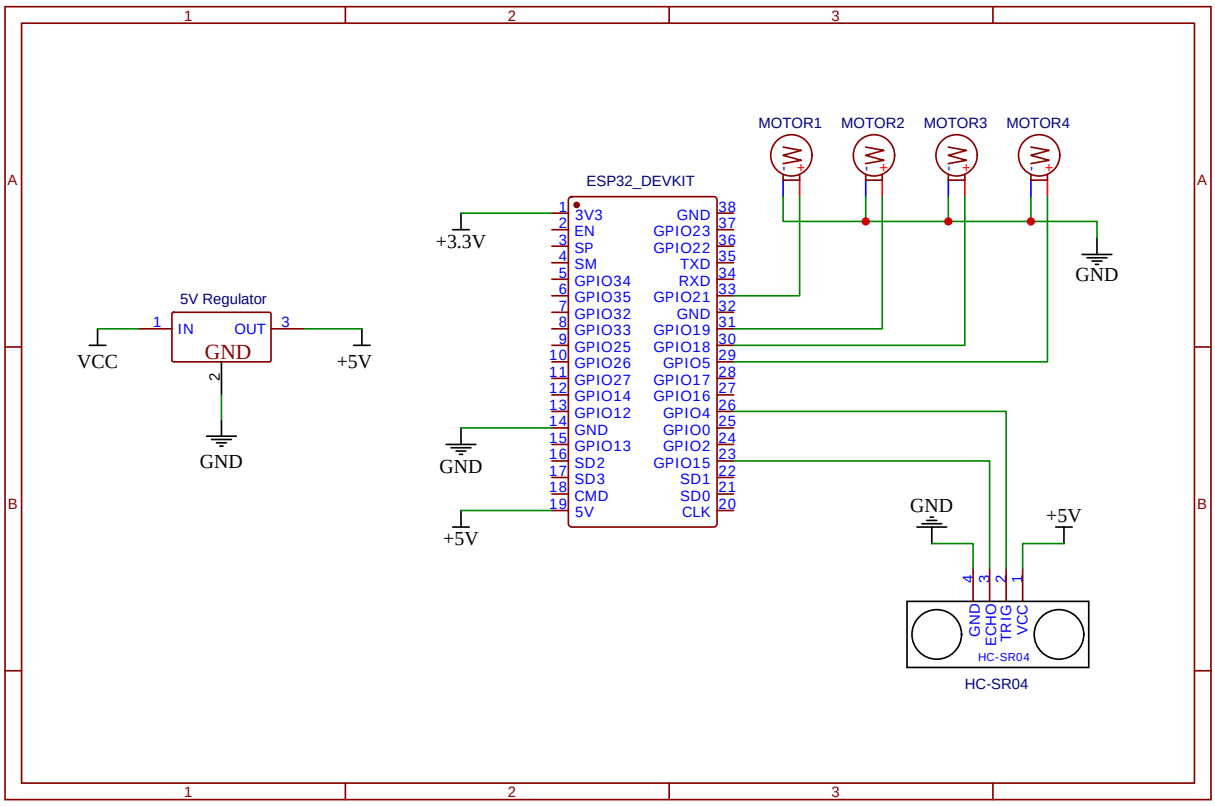

Figure 1: Wireless paddle schematic

3.2 Cross Device Communication

3.2.1 ESP-NOW and Serial I/O

ESP-NOW is a wireless communication protocol developed by Espressif Systems. It supports automatic connection/reconnection, which is a big part of user experience for connected the paddle and dongle. Our boards identify each other through their MAC addresses and begin to send data to each other as soon as they receive power. Since serial I/O is buffered, we must clear the buffer as soon as any connections are made to eliminate old data. Additionally, to optimize game performance, we only read from the serial buffer if it is not empty, rather than waiting for data (which would limit the game loop to the speed at which data is sent).

3.2.2 Data Serialization and Protocol Buffers

We package and serialize data through the use of Google's Protocol Buffers. This greatly reduces any repeated code by avoiding the need to implement data schemas in different languages. Protocol buffers allow for one "proto" file describing our data types, which

can then be used to automatically generate code for encoding/decoding. Data that is serialized from protocol buffers is then encoded into hex to be transmitted. The decoding process decodes the hex data and deserializes the returned bytes into the intended data types described in the "proto" file, see figure 2.

```
message PaddleOut {
    double distance = 1;
    bool buttonClicked = 2;
}

message PaddleIn {
    message Buzz {
        repeated int32 motors = 1;
        int32 durationMillis = 2;
    }
    Buzz buzz = 1;
}
```

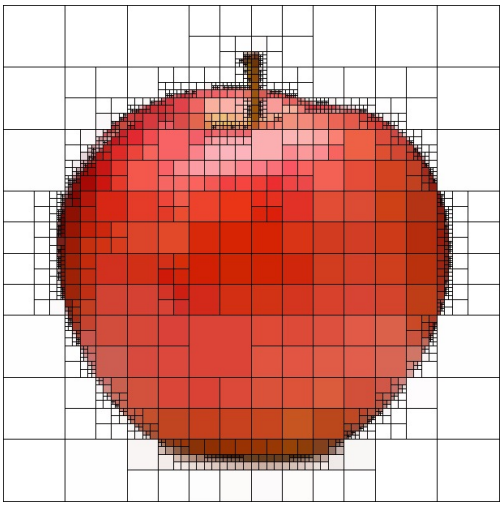

Figure 2: Proto types (left) and an apple with the QuadTree algorithm applied (right)

3.3 Computer Vision

One of our goals for this project was to implement a system which could take any image, drawn or downloaded from the internet, and convert it into a playable image in the game. This would provide an immersive feel for the player and it would be a creative way for us to bridge our hardware and software together in this project. The solution to the former case was to use computer vision to detect shapes in an image. This was accomplished through the utilization of the OpenCV library, a powerful open-source real-time computer vision library originally developed by Intel. This library was especially useful for our project as it not only provided feature detection for our images, but it outputted these features as an array of coordinates (pixels) which is the data that our game needed in order to interpret any image as bricks. In essence, this process was a for loop in which each iteration added a new set of coordinates to a two dimensional Numpy array. A challenge we encountered was interpreting this set of points as shapes. For example, if we passed six random coordinates to the game, how would the game know which points to draw a line between? This was solved by our two dimensional Numpy array as each array of coordinates (shape) given to us through each iteration of the for loop was stored as a separate array of points within a large array of shapes. Each set of coordinate points in a shape array is sorted in the order that the points should be connected, so that the game can consistently interpret and draw polygons of any complexity.

3.4 QuadTree

A simple, straightforward method of converting a coloured image into a brick arrangement would be to pixelate the image into equally sized squares. However, this approach doesn't provide the most compelling results, as either the bricks are all very small, or the bricks are too large to discern the original image. So, we borrowed an idea from an image compression algorithm that uses a quadtree data structure. A quadtree is the same as any tree, except with each node having exactly four children—and these children represent the four quadrants of a square. The image compression algorithm starts off with a single large square encompassing the entire image, with its colour determined by the average colour of all the pixels contained within the square. It then partitions that square into quadrants, where each quadrant is then treated in the same manner as the parent square. We specify an error value, and when the difference between the average colour of the square and the individual pixels within it is less than that error threshold, the algorithm halts the subdividing of the square. As such, the algorithm spits out a series of squares of varying sizes. The larger squares cover areas of the original image with a roughly homogenous colour distribution, while smaller squares encompass regions with finer detail. Figure 2 illustrates the input and output for our quadtree brick creation algorithm.

3.5 Polygon-Ball Collision Detection

The QuadTree image to map generation spits out squares, which are easy to detect collision with a ball (simply check if the distance from the center of the ball to any edge is less than or equal to the radius). However, the OpenCV method of converting a drawing to a map outputs oddly shaped polygons which require a bit more calculation to detect collision with the ball. Note that to collide with a polygon is equivalent to colliding with any of its edges, which connect adjacent vertices. Internally, the game stores shapes as a collection of coordinate points that represent the vertices, so all we need to do is traverse the points in adjacent pairs and see if the ball collides with any of these lines. For the ball to collide with a line, it must be within the "scope" of the line (the angle formed by the vectors from the end points of the line and the ball's center and the endpoints must be acute) and the perpendicular between the ball and the line must be less than or equal to the radius. The reflection axis after collision is determined by the perpendicular vector. To make these computations, a custom Vector2D class was developed.

3.6 Geohashing

It turns out that the above method of collision detection is rather computationally expensive. When it was first implemented, we naively checked if the ball collided with any of the edges for all of the brick polygons. In doing so, the framerate dropped from 200+ frames per second to a mere 15 frames per second. However, we came to realize that since the bricks

are stationary, we could precompute which bricks the ball could possibly be colliding with given its position. To achieve this, we split the grid into 64 regions (to form an 8x8 grid) and group the bricks within this region together. Then, only if the ball is within a region do we check to see if it collides with any of the bricks in that region. This process is known as geohashing and is used by major software products, such as Google Maps to effectively load content. With this approach, our framerate was increased to the 200+ frames per second range.

3.7 Ball Guidance with Paddle

When the ball bounces off of the bricks, it should follow a natural reflection about the axis of reflection. However, to give the user more control in the game, we made it so that depending on the ball's collision point with the paddle, its reflection angle will have a transformation applied to it. To compute the transformation, we used the standard matrix for a counterclockwise rotation that we learned in Math 115.

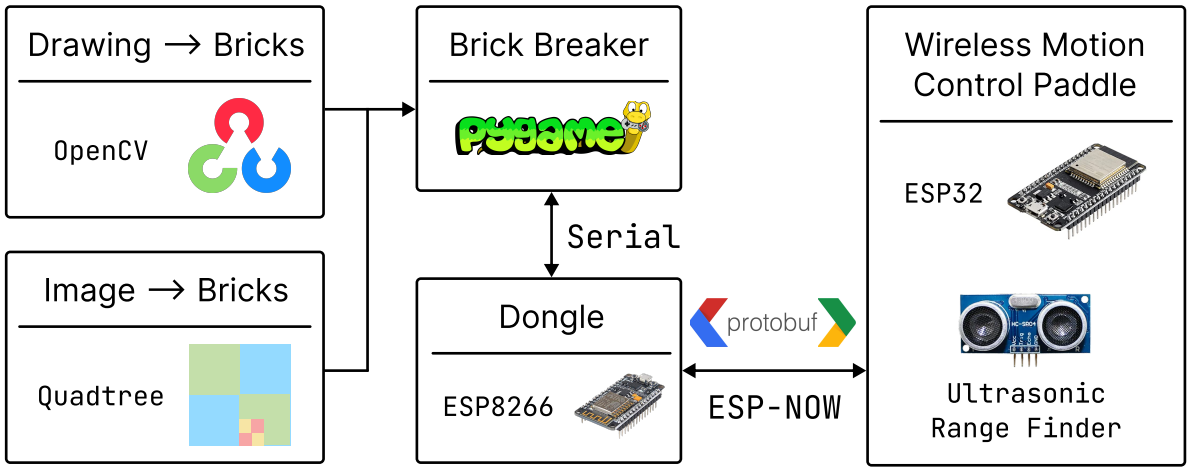

Figure 3: Integration of Hardware and Software

4 Contribution of Group Members

Michael and Hussein primarily worked on the hardware-side of the project and the embedded code, integrating Adam, Emerson, and Muneeb's software so each of our contributions could work together simultaneously as a final product.

Adam was responsible for the computer vision implementation, which mainly consisted of developing the algorithm which converts an image into blocks for the Brick Breaker game.

Muneeb and Emerson were in charge of game implementation and design, ensuring that the game was bug-free, optimized and fun for the user.

Over the period of the 2 months, we met once per two weeks in person to work together and discussed any changes we may need to make if a new idea or problem arised. We also kept in touch regularly via WhatsApp and GitHub, to make communications of any new code/files/changes that were made throughout the process. Although our team was divided into roles, we always helped each other out whenever we were stuck with code implementation or if we needed advice on how to accomplish a given task. We also had weekly meetings with Sara every Tuesday to receive constructive feedback on our progress and asked questions about any of our concerns regarding the project.

5 Final Product Evaluation

Overall, we were able to accomplish the majority of our goals we had planned in the first place. The main part of this project was to create a microcontroller which tracks the user' movements to control the paddle. In addition, we were able to successfully create vibrations, mimicking the feeling of the ball impacting the paddle. Another main part of this game was to develop an algorithm which detects critical points of a drawing given by the user, so that the image will be converted into blocks for the game. This was created successfully since when the ball comes into contact with a block, only that specific part of the image will be removed, rather than the whole image being removed at once.

We did not encounter any glitches or bugs after finalizing our project - not to say that we did not have any during development! For example, the ball was able to stay within the bounds of the screen, the collision of the ball with a block caused the ball to bounce off in the correct direction (and not straight down), the ball did not randomly cut through any blocks, and the ball was able to remove a block everytime it made contact with it. For these reasons, the plans we came up with prior to our project were feasible, which led to us finishing the project within the two months we were given. As with many projects, we can always add more features, especially since we kept this in mind. More on future work is outlined below.

We are very happy with the final result, a fully wireless, rechargeable paddle with a range of 15+ meters, a dongle, and an optimized game with accurate physics. We intended to document and compile all of our research and progress into this report, as well as our GitHub organization.

The only goal we were unable to accomplish within this period of time was creating a server for automatic transfer of a user's picture into the game, where it would then be processed by our algorithms.

6 Design Trade-Offs

During the development of the main hardware, we noticed a problem with the voltages that we were providing to each of the components. Specifically, the ultrasonic distance sensor operates on 5 volts, but we were providing it with power through the 3.3 volt ESP32. We did not have a 5 volt regulator readily available, so we decided to use the Arduino Nano's built in voltage regulator as such. Although it is not a permanent solution, in terms of efficiency and size, it was a good solution to a problem that would have halted our development process for a while.

Additionally, our early plans detailed the use of a gyroscope and accelerometer module, which would have introduced much more mathematical complexity beyond our scope. In the future work section, we detail a few solutions to this problem.

Although the ultrasonic sensor is a great alternative to an accelerometer, it only provides one dimension of displacement data, compared to the 3 dimensional capability of the accelerometer. Moreover, the ultrasonic sensor constrains the playing field, as it requires a solid, flat surface to bounce ultrasonic waves off of. In contrast, a gyroscope-accelerometer position system would allow for the playing of the game anywhere.

7 Future Work

When building our product, we planned on making additions in the future. As such, we ensured that our designs were capable of being upgraded with ease. For example, we designed a slot for a button to be fixed in the case. A button is not currently a feature, but may be useful in the future. The modularity of the case design allows us to further expand on this project in the future if we wish. On the software side, using protocol buffers means that we have automatic serializing/deserializing to types, which allows for backwards compatibility if any changes need to be made to the data schema. A button press event can easily be registered by adding an extra field to the paddleOut type and some extra code on the game side, due to the protocol buffer design choice (rather than conventional JSON or custom serialization methods).

Here is a list of things we would like to do in the future to enhance our product:

1. Create a title screen and add music. That way, this would have made our game more fun and immersive.
2. Make a game over option. As of right now, the only game over option is when all of the blocks are removed from the screen.
3. Create a server through which a drawn image on a whiteboard can automatically be uploaded to the game as a playable image
4. Create a clearer rendered image for the blocks
5. Juicy haptics (more refined audio, visual, and haptic embellishments)
6. Accelerometer for 3 dimensional displacement
7. Button software implementation

References

[1] Audlrg and Instructables. *How to make an infrared camera with an IR led light*. 2018. URL: https://www.instructables.com/HOW-TO-MAKE-AN-INFRARED-CAMERA-WITH-AN-IR-LED-LIGH/.

[2] Bwiggs. *BWIGGS/Quadtree-art: Quadtree Art W/ Golang*. URL: https://github.com/bwiggs/quadtree-art.

[3] *Dead reckoning*. 2022. URL: https://en.wikipedia.org/wiki/Dead_reckoning.

[4] *Faang system design interview: Design A Location Based Service (yelp, google places)*. 2022. URL: https://www.youtube.com/watch?v=M4lR_Va97cQ.

[5] *How to detect shapes in images in python using opencv?* 2021. URL: https://www.geeksforgeeks.org/how-to-detect-shapes-in-images-in-python-using-opencv/.

[6] Sharon Lin. *Using ir leds to hide in plain sight*. 2020. URL: https://hackaday.com/2020/02/28/using-ir-leds-to-hide-in-plain-sight/.

[7] *OpenCV*. 2022. URL: https://opencv.org/.

[8] Zach Rolfness. *StickyAR*. URL: https://devpost.com/software/stickyar.

[9] Kurt Seifert and Oscar Camacho. *Implementing positioning algorithms using accelerometers - NXP*. URL: https://www.nxp.com/docs/en/application-note/AN3397.pdf.

[10] *Serialiazing your data with Protobuf*. 2019. URL: https://blog.conan.io/2019/03/06/Serializing-your-data-with-Protobuf.html.

[11] Singhal, Tanay and Schneider, Oliver. "Juicy Haptic Design: Vibrotactile Embellishments Can Improve Player Experience in Games". In: UWSpace, 2021. URL: http://hdl.handle.net/10012/16894.