EXTENDED ABSTRACT
Ball collision pong game w/ wall POP IT

By Edvin Frosterud and Yas Asghari

DATE: 2022-03-04

Objective.

This is an ***advanced project***, a game which is inspired by Breakout and Pong, that we call POPIT. The goal in the game is to hit through the 48 bricks in the middle of the screen either against an opponent in 2-Player mode or against an AI with an incrementing highscore on the top of the screen and to make sure that the ball is not missed by the paddle, which would otherwise lead to Game Over.

Requirements & Solution.

For the project to reach Advanced, we needed to implement animations in both x and y directions using the basic I/O shield OLED graphical display. The program interacts with the whole screen pixel by pixel. We write each pixel individually to an array of uint8_t using drawSprite and from there we draw each pixel separately from an array we called displayFULL using write2Oled. write2OLED uses SPI.

For every gameObj we have made a struct that includes the position, an alive boolean, and its velocity. We wanted to include another struct inside the gameObj that would contain the textures width and height, however this did not work. This is because sending through member variables of a struct through a function causes bugs. We could have just added these to gameObj but it would have led to a messier coding.

Furthermore, we implemented 2-player mode and 1-player mode against AI with difficulty levels that are controlled with a gameSpeed modifier that increases all movement correspondingly.

Collision is handled using velocity and the current position of each gameObj on the screen. We look at the current trajectory before impact and make sure it does not hit any bricks or paddles. If it does, we invert its current trajectory before we add the velocity to the current position. If it hits the paddle, we use different calculations in a separate function called paddleCheck. Depending on how far away from the middle you hit the paddle, the angle of the bounce is changed accordingly.

We use the same collision code for the detecting collision in the bricks and paddles but the bounce is calculated differently in order to have the ball hit the paddle from the y direction. It is

important to note that our x and y directions are swapped in order to make more space for potential game play which makes the y direction x for this game.

High score is saved in a typedefined struct called scoreboard that contains the top3 numbers and top3 names corresponding to each score. After Game Over, we check if the score has surpassed the scores in the leaderboards, then we check if you have surpassed other scores. We then write and move down the leaderboard correspondingly.

Verification & Reflections.

We faced many bugs at the end of the project. In order to solve each compilation within each function that we wrote, we decided to print text messages corresponding to where we would get stuck in the code in order to not have to go through the entire function every time manually.

Collision had most bugs, since the first function we wrote was drawSprite, and it turns out our sprites were being mirrored across the y axis. This meant that the ball texture, that we had oriented to the top left, was being instead rendered to the right of the hitbox of the ball, which led to a lot of confusion.

After a lot of deliberating and trying to fix the issue, we just decided to fix it by changing the sprite to take that into account. Since all our other sprites were a full 8 pixels wide, this was never anything we noticed. Mirroring a 1 thick layer of pixels doesn't change the look or hitbox of the pixels. It was only because we drew the ball incorrectly that a lot of our bugs, for example our paddle collision missing sometimes, the ball getting stuck on the paddle, and simply not rendering at the edge of the screen.

To verify this problem, we basically just made a sprite where we could identify the different corners and figure out what parts of the sprite were being moved in what direction.

To verify for other bugs, like menu, gameplay, etc, we basically just played the game until we found a bug, which was inevitable, and then tried to figure out why that bug was happening in particular by slowing the game down and trying to replicate the situation by spawning the balls in similar situations.

For example, to make sure that the spot position in the sorting of the leaderboard for the scoreboard is working, we printed a number corresponding to the spot the program placed the user in when their score was higher than the previous ones and made sure that the function for sorting worked.

We also noted some smaller visual glitches, for example printing a string with Z involved made it invisible. This was because we were using ‘Z’ > x instead of less than or equal.

Contributions.

We did 90% together, using pair programming for the programming parts. It was all done in school, together using both our brains and computing knowledge. Collision calculations for the brick were heavily inspired by Edvin Frosteruds precious C# project, which also later helped with paddleCheck function as well as boundary calculations. MENU and High score was brainstormed and implemented by Yas Asghari, and AI and 2 player mode by Edvin Frosterud and drawbitmap, vectors and textures together with drawings to make sure both of us have the design of the game the way we both would like.