ĐẠI HỌC BÁCH KHOA HÀ NỘI

BÁO CÁO BÀI TẬP LỚN

Đề tài: Xây dựng trò chơi bắn trứng khủng long trên kit phát triển STM32F429ZIT6

Nhóm 4

Hoàng Nhật Minh	20215426
Phạm Đức Minh	20210586
Đào Duy Anh	20210041
Nguyễn Đức Hiếu	20215370

Giảng viên hướng dẫn: TS. Ngô Lam Trung _____

Chữ kí GVHD

Môn: Hệ nhúng

Trường: Công nghệ Thông tin và Truyền thông

HÀ NỘI, 1/2025

MỤC LỤC

DANH MỤC HÌNH VẼ

CHƯƠNG 1. GIỚI THIỆU

1.1 Mô tả đề tài

Tựa game bắn trứng khủng long (tên tiếng Anh: Dynomite!) là một trò chơi giải đố tính điểm kinh điển của hãng Popcap. Cách chơi của tựa game rất là đơn giản, trong suốt quá trình chơi, những quả trứng đủ sắc màu liên tục rơi xuống phía dưới và nhiệm vụ của người chơi là bắn nổ chúng trước khi có bất cứ quả trứng nào rơi xuống đè bẹp người chơi. Để bắn nổ trứng, người chơi được cung cấp một súng cao su với đạn dược là các quả trứng được cung cấp một cách ngẫu nhiên từ trò chơi. Khi quả trứng của người chơi được bắn lên va chạm với các quả trứng đang rơi tạo ra một dãy (tối thiểu ba) quả trứng cùng màu thì các quả cùng màu đó sẽ phát nổ và người chơi sẽ được điểm thưởng.

Hình 1.1: Tựa game Dynomite! kinh điển

Với mô tả trên, ta có thể thấy game bắn trứng khủng long có hệ thống không quá phức tạp, nhưng lối chơi lại rất lôi cuốn. Với các đặc điểm đó, việc có thể xây dựng trò chơi bắn trứng khủng long trên kit phát triển STM32F4 là hoàn toàn khả thi. Mục tiêu của đề tài này của nhóm là xây dựng tựa game dynomite! trên kit STM32F4 với các tính năng cơ bản nhất của trò chơi.

1.2 Phân công nhiệm vụ

Thành viên	Công việc
Hoàng Nhật Minh	Xử lý âm thanh
Đào Duy Anh	Xử lý tạo màn chơi và sinh ngẫu nhiên
Phạm Đức Minh	Xử lý game engine, va chạm vật lý và tích hợp
Nguyễn Đức Hiếu	Mua hàng, lắp mạch, kiểm thử

CHƯƠNG 2. HỆ THỐNG PHẦN CỨNG

2.1 Kit STM32F429ZIT6

Nhóm sử dụng kit phát triển STM32F429ZIT6 được cung cấp, hướng dẫn sử dụng trên phòng học lý thuyết, thực hành cho việc xử lý các tác vụ khác nhau trên hệ thống máy chơi game bắn trứng khủng long bằng tay cầm.

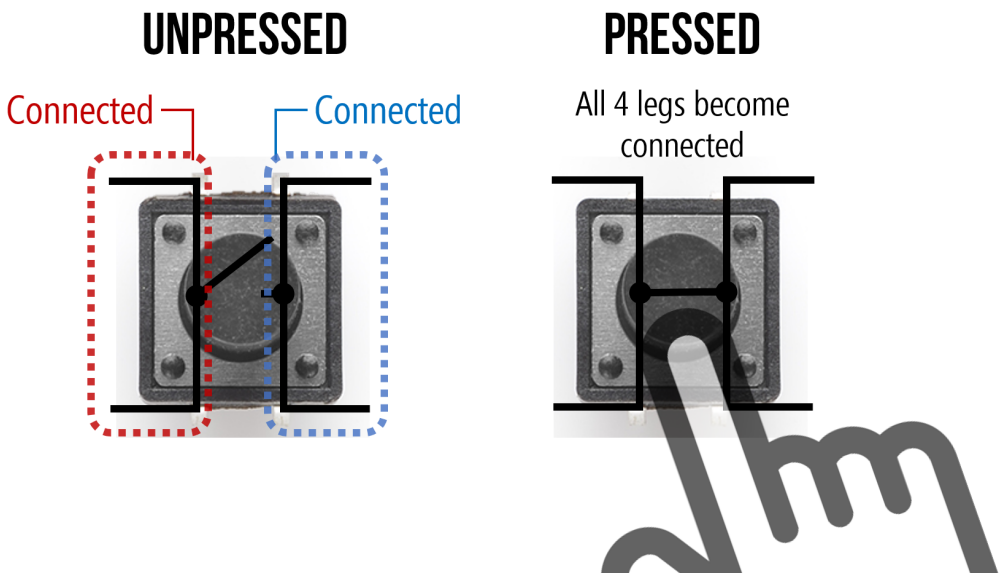
2.2 Nút bấm

2.2.1 Cấu tạo

- Thân nút: Làm bằng nhựa, là phần bên ngoài để người dùng bấm vào
- Lò xo: Giúp nút trở về ban đầu sau khi nhấn
- Chân kết nối: Gồm 4 chân kim loại, 2 cặp chân đối diện nhau được nối ngầm bên trong mạch, khi ấn nút, các chân sẽ nối lại với nhau

2.2.2 Nguyên lý hoạt động

- Trạng thái không nhấn: Khi nút không nhấn, các tiếp điểm không chạm vào nhau, mạch điện hở và không có dòng điện chạy qua.
- Trạng thái nhấn: Khi nhấn nút, lực tác động làm cho các tiếp điểm chạm vào nhau, mạch điện được đóng lại, hành động được điều khiển bởi nút bấm sẽ hoạt động.

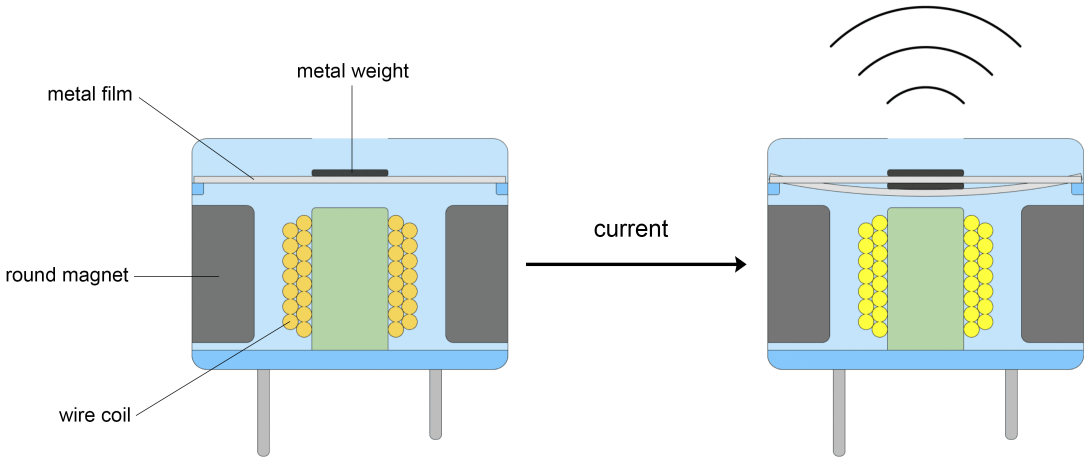


Hình 2.1: Cấu tạo nút bấm

2.3 Còi buzzer

Còi buzzer 3V là thiết bị phát ra âm thanh, thường được sử dụng trong báo động, thông báo, hoặc tín hiệu âm thanh trong các thiết bị điện tử. Còi buzzer của nhóm là loại bị động (passive): Điện áp cố định và cần một tín hiệu điều khiển (PWM) để phát ra âm thanh với tần số và cao độ khác nhau.

1. Cấu tạo
    - Màng rung: Bộ phận chính tạo ra âm thanh, làm bằng vật liệu nhẹ và linh hoạt như nhựa hoặc kim loại mỏng.
    - Cuộn dây: Là cuộn dây đồng cuốn quanh lõi từ tính.
    - Nam châm: Được đặt cố định trong buzzer.
    - Chân kết nối: Gồm một chân dương và một chân âm.



Hình 2.2: Cấu tạo còi buzzer

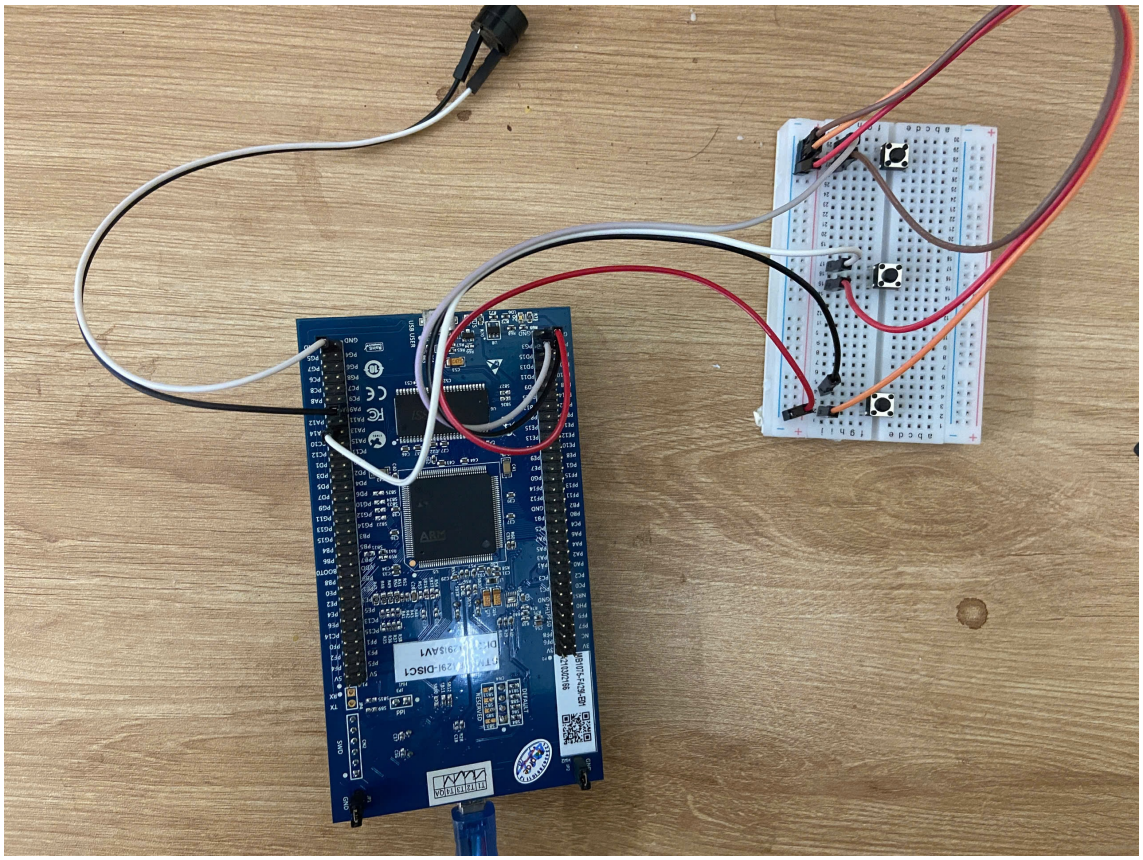
2. Nguyên lý hoạt động
    - Khi cấp tín hiệu PWM vào buzzer, dòng điện sẽ chạy qua cuộn dây.
    - Từ trường sinh ra trong cuộn dây theo nguyên lý cảm ứng điện từ sẽ tạo ra lực đẩy hoặc kéo màng rung, khiến nó dao động.
    - Màng rung dao động, tạo ra sóng âm thanh trong không khí.
    - Âm thanh phát ra qua các lỗ thoát âm trên vỏ buzzer.

2.4 Mô tả sơ đồ mạch hệ thống

1. Vi điều khiển STM32f429ZIT6 được xử dụng làm bộ xử lý trung tâm của hệ thống.
2. Buzzer 3V:

- Cực dương nối với chân PA9 của kit, sử dụng phương pháp PWM để tạo ra âm thanh với các tần số và độ rộng xung khác nhau.
- Cực âm được nối với GND.

3. Nút bấm: 3 nút bấm được sử dụng, với chân Output lần lượt được nối với các chân PG2, PA14, PG3 tương ứng với hành động sang trái, sang phải và bắn trứng. 3 chân GND của 3 nút bấm được nối với GND của mạch



Hình 2.3: Sơ đồ mạch

CHƯƠNG 3. MÔ TẢ GIẢI PHÁP

3.1 Hệ thống sinh số ngẫu nhiên

3.1.1 Bản chất RNG trong STM32

Hệ thống RNG trên STM32 sử dụng các thành phần mạch analog và kỹ thuật xử lý số để tạo ra các giá trị ngẫu nhiên.

- Sinh nhiễu analog: Nhiễu (*noise*) được sinh ra từ mạch analog. Đây là loại tín hiệu không thể dự đoán được, tạo nền tảng cho tính ngẫu nhiên của RNG.
- Bộ dao động vòng (*Ring Oscillators*): Mạch analog được cấu tạo từ các *ring oscillators*, là các mạch dao động tạo ra tín hiệu không ổn định. Những biến đổi ngẫu nhiên trong điện áp, dòng điện hoặc nhiệt độ khiến đầu ra của các dao động này thay đổi theo cách không dự đoán được. Đầu ra từ các dao động được đưa qua cổng XOR (*Exclusive OR*), tạo ra tín hiệu hỗn hợp phi tuyến tính, tăng cường tính ngẫu nhiên.
- Bộ dịch chuyển phản hồi tuyến tính (LFSR): Tín hiệu từ mạch analog được chuyển đến *Linear Feedback Shift Register (LFSR)*. Bộ này thực hiện chuyển đổi tín hiệu đầu vào thành chuỗi bit nhị phân, đảm bảo tín hiệu có tính ngẫu nhiên cao và ở dạng dữ liệu số.
- Lưu trữ số ngẫu nhiên: Số ngẫu nhiên 32-bit sau khi xử lý được lưu trong thanh ghi *Data Register (DR)* tại địa chỉ: `RNG->DR` hoặc `0x50060808`.

3.1.2 Sử dụng

- Khởi tạo RNG (`TM_RNG_Init`):
  - Bật đồng hồ cho mô-đun RNG bằng cách bật bit `RNGEN` trong thanh ghi `AHB2ENR` của RCC.
  - Kích hoạt mô-đun RNG bằng cách thiết lập bit `RNGEN` trong thanh ghi `CR` của RNG.

```
1 void TM_RNG_Init(void) {
2     /* Bật đồng hồ */
3     RCC->AHB2ENR |= RCC_AHB2ENR_RNGEN;
4 
5     /* Bật bộ RNG */
6     RNG->CR |= RNG_CR_RNGEN;
7 }
```

- Đọc số ngẫu nhiên (`TM_RNG_Get`):

– Kiểm tra trạng thái số ngẫu nhiên thông qua bit `DRDY` trong thanh ghi trạng thái `SR`.

– Nếu số ngẫu nhiên đã sẵn sàng, đọc giá trị từ thanh ghi `DR`.

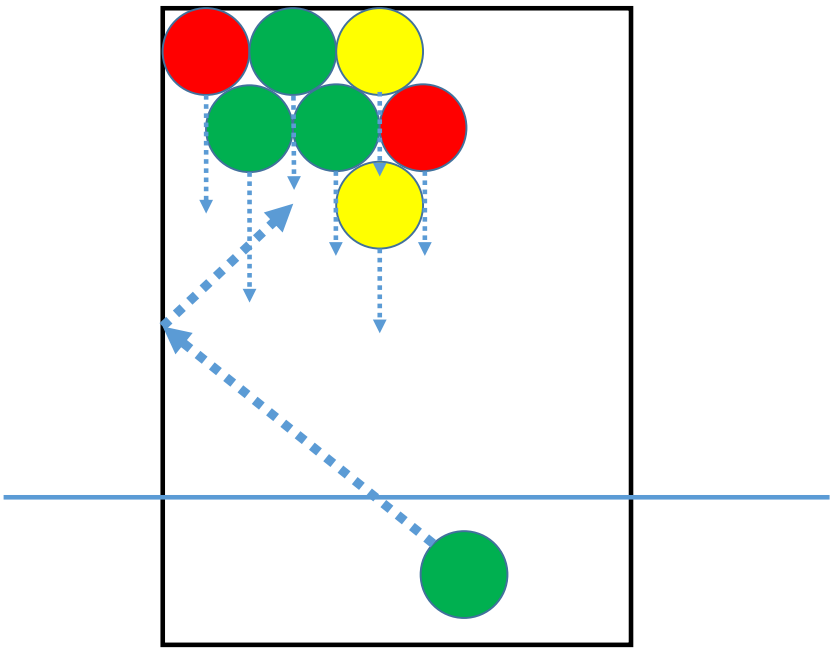
```
uint32_t TM_RNG_Get(void) {
    /* Đợi tới khi số ngẫu nhiên được sinh ra */
    while(!(RNG->SR & (RNG_SR_DRDY)));

    /* Lấy số ngẫu nhiên đã được sinh */
    return RNG->DR;
}
```

3.1.3 Ứng dụng trong hệ thống

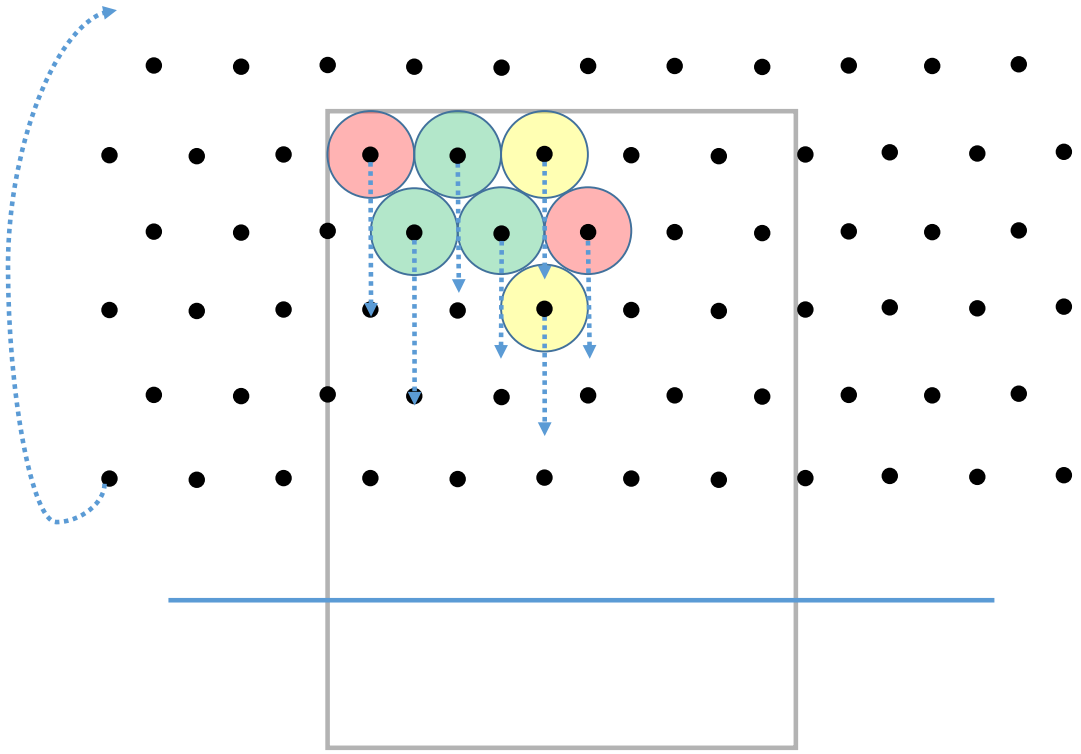
Phương pháp sinh ngẫu nhiên này được sử dụng để tạo màu sắc cho các quả trứng trên lưới trong mục 3.2.1. Điều này giúp đảm bảo các màn chơi được thiết kế với tính ngẫu nhiên cao, tăng trải nghiệm và sự thú vị cho người chơi.

3.2 Engine của trò chơi



Hình 3.1: Minh họa engine trò chơi

Mục 3.2 này của báo cáo sẽ trình bày một cách tổng quát phương hướng phát triển engine đã được xây dựng trong trò chơi. Để dễ dàng phân tích engine của trò



Hình 3.2: Minh họa về grid của màn chơi. Mũi tên nét đứt thể hiện tính năng trứng rơi theo thời gian.

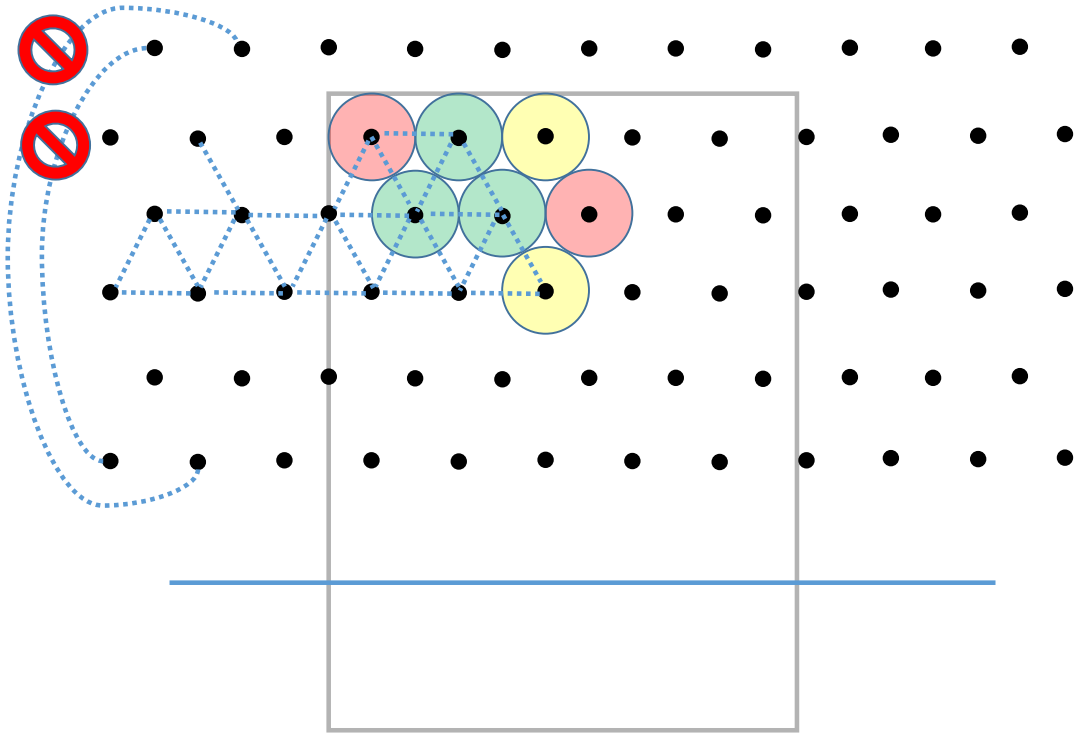
chơi bắn trứng khủng long được xây dựng trên kit phát triển STM32, ta sẽ chia hệ thống engine (minh họa hình 3.1) ra thành ba thành phần chính: phần sinh màn chơi trứng rơi xuống phần vật lý bắn trứng, và phần nổ trứng - tính năng tính điểm của trò chơi.

3.2.1 Trứng rơi

Thành phần màn chơi của game được định nghĩa có dạng là một lưới grid kích thước $H \times W$ (hình 3.2). Trong đó, các điểm (i, j), với $0 \leq i < H$, $0 \leq j < W$, trên grid được khởi tạo với tọa độ hình học bằng

$$\begin{cases} (L_x + 2r \cdot j + r, L_y + 2r \cdot i + r) & \text{nếu } i \text{ chẵn} \\ (L_x + 2r \cdot j + r, L_y + 2r \cdot i + 2r) & \text{nếu } i \text{ lẻ} \end{cases} \tag{3.1}$$

, trong đó L_x, L_y là biên trái, biên phải của lưới và r là bán kính của các quả trứng. Khác biệt tọa độ của các điểm giữa hàng chẵn và hàng hàng lẻ trên lưới là các điểm trên hàng lẻ sẽ bị dịch đi r đơn vị về bên phải so với các điểm ở hàng chẵn. Ngoài ra, về mặt mô hình hóa thì giá trị của L_x, L_y có thể được đặt tùy ý sao cho grid bao phủ hết toàn bộ màn hình. Nhưng để tiết kiệm bộ nhớ và chi phí tính toán, ta sẽ tạo

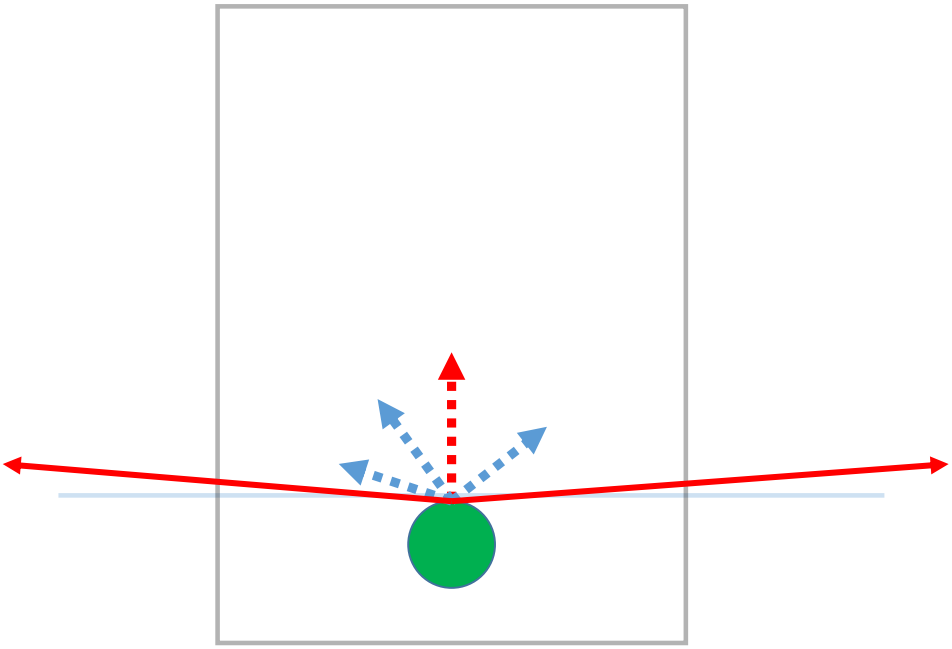


Hình 3.3: Minh họa cấu trúc liên kết của lưới

lưới khớp với màn hình nhất có thể, hay khởi tạo L_x, L_y bằng các giá trị gần 0 nhất có thể.

Các quả trứng được sinh ra sẽ có tâm ứng với các điểm trên lưới. Tính năng các quả trứng rơi xuống tương ứng với việc tăng tọa độ trục tung của các điểm trên grid lên một đơn vị theo thời gian. Khi một hàng của lưới có tọa độ trục tung bị rơi xuống một ngưỡng nhất định, ta sẽ đẩy hàng đó quay ngược lên trên để trở thành hàng đầu của lưới bằng cách gán tọa độ trục tung của các điểm trên hàng đó bằng L_y. Ở đây, ta có thể hiểu là các điểm trên grid rơi xuống theo dạng xoay vòng từ trên xuống. Sau khi hàng mới được đẩy lên đầu, ta sẽ thực hiện sinh trứng với màu sắc ngẫu nhiên tại các điểm ở hàng đó. Ở đây ta có thể giả thiết rằng mọi điểm của hàng mới này đều không có trứng và thực hiện sinh trứng mới cho cả hàng.

Tiếp theo, ta sẽ định nghĩa cấu trúc liên kết (hai chiều) giữa các điểm trên lưới để phục vụ phần nổ trứng phía sau, mục 3.2.3. Đầu tiên là các điểm kề nhau trên một hàng đều có cạnh kết nối. Nói một cách khác điểm (i, j) có liên kết tới điểm $(i, j + 1)$, với $0 \leq i < H$, $0 \leq j < W - 1$. Ngoài ra giữa các hàng cũng có liên kết với nhau, cụ thể là điểm (i, j) $(0 \leq i < H, 0 \leq j < W - 1)$ với i lẻ sẽ có liên kết tới bốn điểm lân cận: $((i - 1) \bmod H, j)$, $((i - 1) \bmod H, j + 1)$, $((i + 1) \bmod H, j)$, $((i + 1) \bmod H, j + 1$. Tuy nhiên, ở đây ta có một ngoại lệ là sẽ không có liên kết



Hình 3.4: Minh họa tính năng bắn trứng

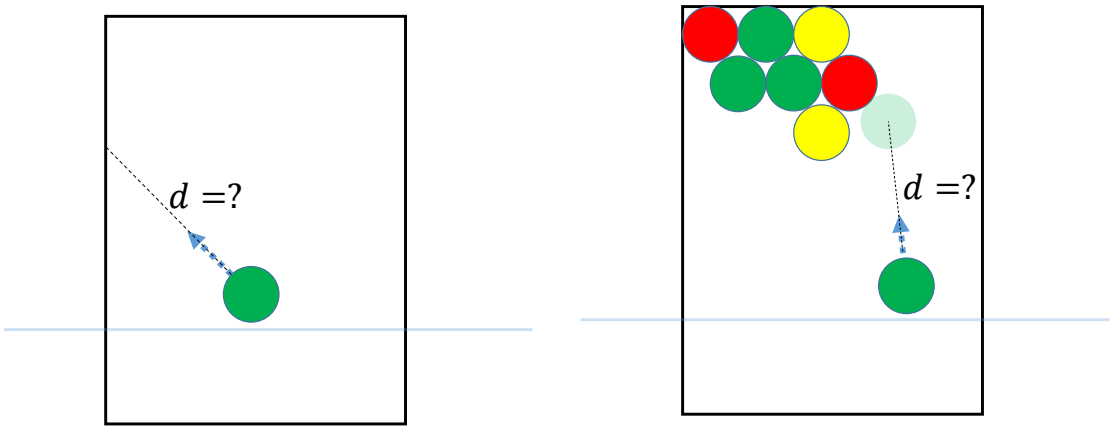
giữa hàng cuối cùng và hàng đầu tiên (xét về tọa độ hình học) trên grid, như minh họa trong hình 3.3.

Như vậy, để duy trì tính năng rơi trứng, hệ thống cần tiêu tốn tài nguyên tính toán với độ phức tạp bằng $O(H \times W)$.

3.2.2 Bắn trứng

Các quả trứng được cung cấp cho người chơi được sinh ra với màu sắc ngẫu nhiên khác nhau tại một ví trị cố định. Ban đầu, hướng bắn của quả trứng sẽ là cố định hướng lên trên màn hình, hay véc-tơ bắn bằng $\vec{v} = (0, -1)$. Việc thay đổi góc bắn có thể được thực hiện bởi người chơi, khi đó véc-tơ $\vec{v}$ sẽ được xoay để ứng với với sự thay đổi. Ngoài ra, hệ thống sẽ giới hạn góc bắn của người chơi chỉ được về phía trên của màn hình (minh họa bằng mũi tên đỏ liền trong hình 3.4). Khi người chơi thực hiện việc bắn trứng đi, quả trứng sẽ di chuyển theo hướng $\vec{v}$. Sau mỗi sự kiện thời gian (tick event) trong trò chơi, quả trứng đó sẽ bay với tổng độ dài quãng đường tối đa ứng với tốc độ của quả bóng.

Tiếp theo, ta cần xem xét tới tính năng va chạm của trứng trong trò chơi. Hệ thống trò chơi này gồm hai loại va chạm: va chạm với biên của màn hình và va chạm với các quả trứng đang rơi. Để xác định xem quả trứng được bắn ra đang



(a) Xác định khoảng cách va chạm tường **(b)** Xác định khoảng cách va chạm với trứng

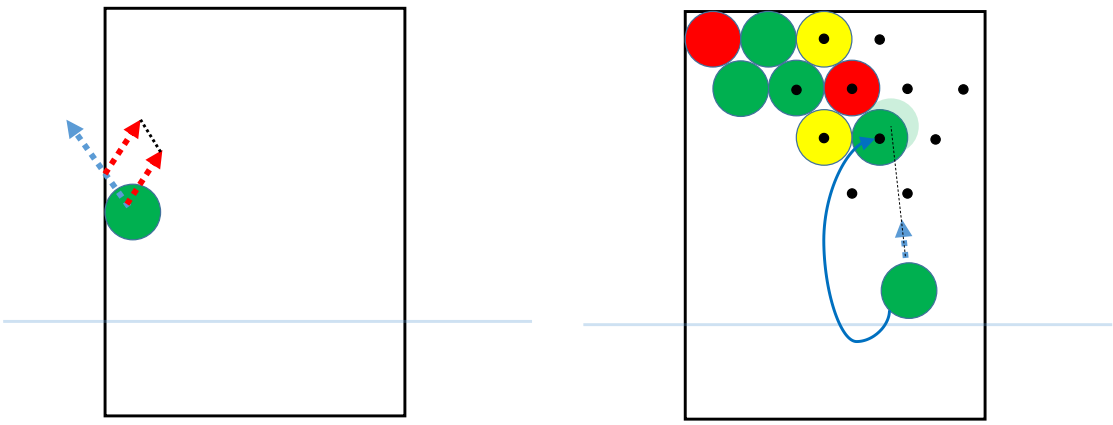
Hình 3.5: Minh họa việc xác định va chạm

bay theo hướng $\vec{v}$ có va chạm vào tường hoặc vào các quả trứng khác đang rơi hay không thì ta cần xác định khoảng cách d giữa quả trứng bắn ra với các vật thể đó theo véc-tơ $\vec{v}$ như minh họa trong hình 3.6. Việc xác định khoảng cách va chạm với biên của màn hình khá là đơn giản vì việc đó tương đương với giải phương trình bậc nhất, với độ phức tạp tính toán $O(1)$. Đối với va chạm với các quả trứng đang rơi, ta sẽ cần giải phương trình bậc hai, đi tìm thời gian mà khoảng cách giữa tâm của hai quả trứng bằng $2r$ khi quả trứng đang bay theo hướng $\vec{v}$. Do ta sử dụng kiểu `float` để mã hóa số thực nên hai quả trứng được coi là va chạm nhau nếu khoảng cách giữa tâm của chúng nhỏ hơn hoặc bằng $2r + \epsilon$ (giá trị epsilon $\epsilon = 10^{-4}$ được chọn để cài đặt trong chương trình). Ở đây, vì phải xác định khoảng cách va chạm với từng quả trứng trên lưới nên tổng độ phức tạp của việc xác định va chạm sẽ là $O(H \times W)$

Khi va chạm sẽ có hai loại sự kiện xảy ra ứng với mỗi loại va chạm. Nếu quả trứng được bắn ra theo hướng $\vec{v}$ va chạm với tường, thì nó sẽ chuyển hướng bay theo hướng $\vec{v'}$ (mũi tên đỏ trong hình 3.6a), trong đó $\vec{v'}$ là véc-tơ phản xạ của $\vec{v}$. Ngược lại, nếu quả trứng bắn ra va chạm với quả trứng khác trên lưới, thì hệ thống sẽ đẩy quả trứng đó vào điểm trên grid gần nhất mà không có trứng (minh họa hình 3.6a). Và sau đó là thực hiện sự kiện nổ trứng (mục 3.2.3), nếu có.

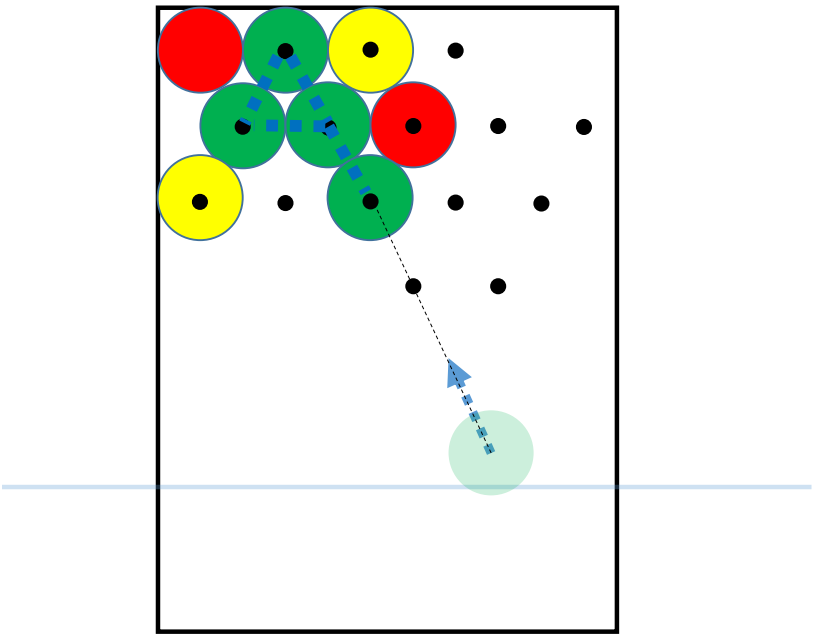
3.2.3 Nổ trứng

Khi quả trứng được bắn ra va chạm với trứng ở trên lưới, hệ thống cần xác định các quả trứng cùng màu có cùng thành phần liên thông với cấu trúc liên kết được định nghĩa trong mục 3.2.1. Hình 3.7 minh họa thành phần liên thông cần xác định là các đường màu xanh dương nét đứt nối tâm của các quả trứng màu xanh lại với

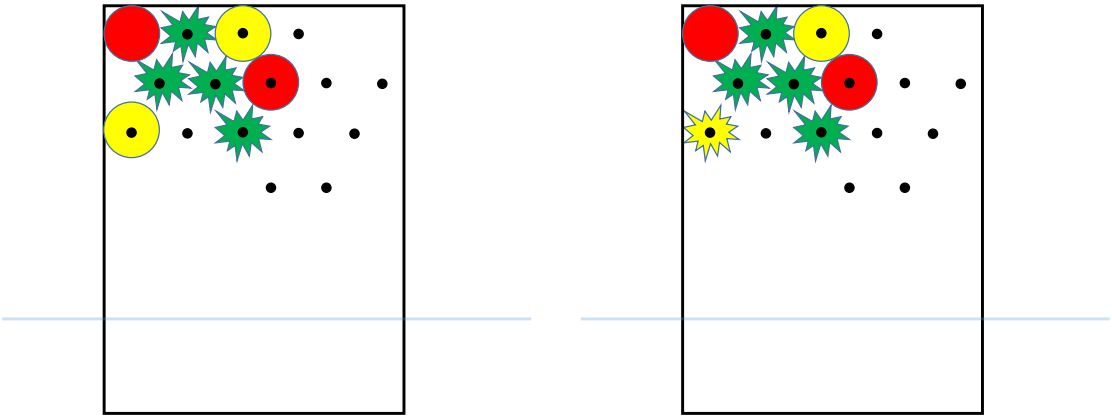


(a) Trứng bắn ra va chạm tường **(b)** Trứng bắn ra va chạm vào trứng trên lưới

Hình 3.6: Minh họa sự kiện xảy ra khi va chạm



Hình 3.7: Minh họa xác định thành phần liên thông



(a) Nổ các quả trứng cùng màu **(b)** Nổ các quả trứng bị lơ lửng

Hình 3.8: Minh họa sự kiện nổ trứng

nhau. Việc xác định này có thể dễ dàng được thực hiện bằng thuật toán BFS từ điểm trên lưới mà quả trứng bắn ra được hệ thống đẩy vào. Sau đó, các điểm thuộc grid được đánh dấu thăm sẽ được hệ thống cho phát nổ, xóa sự tồn tại khỏi màn hình trò chơi. Tuy nhiên, sau bước đó, màn chơi có thể xảy ra trường hợp các quả trứng bị lơ lửng trong không trung như quả màu vàng trong hình 3.8a. Lúc này, ta cần phải thực hiện thêm một lần BFS nữa để xác định các quả trứng có liên kết tới hàng có tọa độ trục tung thấp nhất bên trên để xác định các quả trứng bị lơ lửng, rồi thực hiện kích nổ như hình 3.8b.

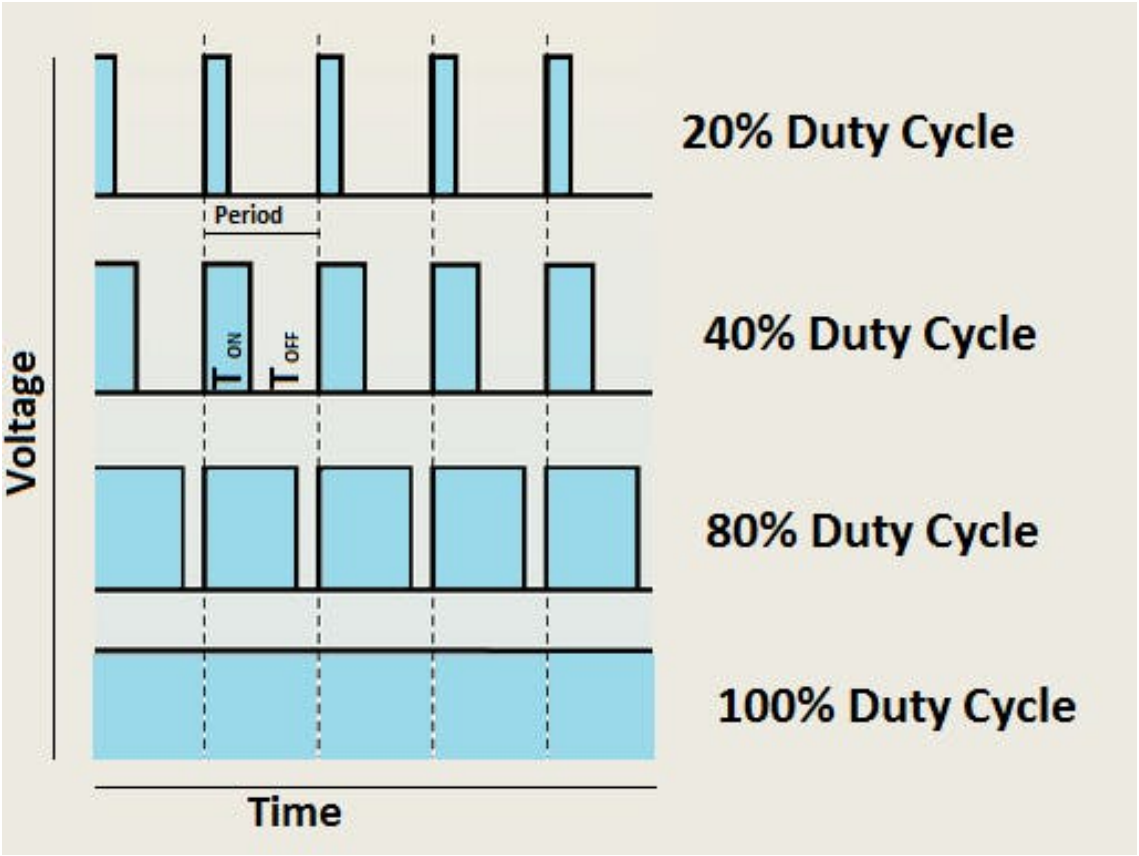
3.3 Tính năng phát nhạc

3.3.1 Lý thuyết

Còi buzzer (passive): Cần tín hiệu PWM để điều khiển tần số âm thanh, khi PWM thay đổi, tín hiệu âm thanh cũng thay đổi theo

PWM (Pulse Width Modulation) - Điều chế độ rộng xung

- PWM là một kỹ thuật điều chế tín hiệu bằng cách thay đổi độ rộng xung trong một chu kỳ cố định.
- Mục đích của PWM là điều khiển tín hiệu âm thanh đầu ra (hoặc công suất) mà không cần thay đổi điện áp đầu vào.
- PWM được sử dụng rộng rãi trong điều khiển động cơ, điều chỉnh độ sáng đèn LED, tạo âm thanh, và nhiều ứng dụng khác.



Hình 3.9: Nguyên lý hoạt động PWM

3.3.2 Cấu hình âm thanh

Sử dụng TIM1_CH2 trên PA9

- Thiết lập Timer TIM1: Clock Source ở chế độ Internal Clock, Channel2 là PWM Generation CH2.
- Chọn chân PA9 là TIM1_CH2 trên file IOC.
- Cấu hình Clock Configuration: Tần số đầu vào bằng 8 MHz, xung nhịp bằng 180 MHz.

Xử lý âm thanh: Quy trình gồm các bước:

1. Chuyển đổi file MIDI thành các nốt nhạc:

   Sử dụng các công cụ chuyển đổi hoặc thư viện để trích xuất thông tin từ file MIDI, vd: Arduino MIDI Converter. Công cụ sẽ chuyển đổi file nhạc MIDI thành các note (tone), khoảng thời gian phát (duration), và quãng nghỉ (delay).

2. Định nghĩa các note nhạc và đoạn nhạc:

   - Mỗi note nhạc có một tần số tương ứng. Ví dụ:

     ```
     #define NOTE_C4  262 // Tần số nốt C4 (262 Hz)
     ```

```
#define NOTE_D4  294 // Tần số nốt D4 (294 Hz)
#define NOTE_E4  330 // Tần số nốt E4 (330 Hz)
#define NOTE_F4  349 // Tần số nốt F4 (349 Hz)
#define NOTE_G4  392 // Tần số nốt G4 (392 Hz)
```

- Tạo một bản nhạc tương ứng với các note và thời gian phát tương ứng.

```
const int openingSound[38][3] = {
  {NOTE_E7, 125, 75},
  {NOTE_E7, 125, 75},
  {0, 125, 125},
  {NOTE_E7, 125, 75},
  {0, 125, 125},
  {NOTE_C7, 125, 75},
  {NOTE_E7, 125, 75},
  {NOTE_G7, 125, 75},
  {0, 125, 125},
  {NOTE_G6, 125, 75},
  {0, 125, 125},
  {NOTE_C7, 125, 75},
  {NOTE_G6, 125, 75},
  {0, 125, 125},
  {NOTE_E6, 125, 75},
  {NOTE_A6, 125, 75},
  {NOTE_B6, 125, 75},
  {NOTE_A6, 125, 75},
  {NOTE_GS6, 125, 75},
  {NOTE_B6, 125, 75},
  {NOTE_E7, 125, 75},
  {NOTE_C7, 125, 75},
  {NOTE_G7, 125, 75},
  {NOTE_A7, 125, 75},
  {NOTE_F7, 125, 75},
  {NOTE_F6, 125, 75},
};
```

3. Cấu hình PWM để phát âm thanh

- Sử dụng STM32CubeMX để cấu hình Timer (ví dụ: TIM2) ở chế độ PWM.
- Thiết lập Prescaler và Counter Period để xác định tần số PWM.
- Thiết lập Duty Cycle.

```
// Khởi tạo Timer
TIM_HandleTypeDef htim2;
htim2.Instance = TIM2;
htim2.Init.Prescaler = 72 - 1; // Chia tần số xuống 1 MHz
htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
htim2.Init.Period = 1000 - 1; // Tần số PWM = 1 kHz
HAL_TIM_PWM_Init(&htim2);

// Cấu hình PWM
TIM_OC_InitTypeDef sConfigOC;
sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 500; // Duty Cycle 50%
HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_

// Bắt đầu PWM
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
```

4. Phát âm thanh trên buzzer

- Hàm `tone`: Hàm này tạo tín hiệu PWM với tần số tương ứng với nốt nhạc.

```
1  void tone(TIM_HandleTypeDef *htim, uint32_t channel,
2          uint16_t frequency, uint16_t duration) {
3          // Chu kỳ của tín hiệu PWM
4      uint32_t period = 1000000 / frequency;
5      // Độ rộng xung (50% duty cycle)
6      uint32_t pulse = period / 2;
7
8      // Cấu hình lại tần số PWM
9      __HAL_TIM_SET_AUTORELOAD(htim, period - 1);
10     __HAL_TIM_SET_COMPARE(htim, channel, pulse);
11
12     // Bật PWM
13     HAL_TIM_PWM_Start(htim, channel);
14
```

```
    // Đợi trong thời gian duration
    HAL_Delay(duration);

    // Tắt PWM sau thời gian duration
    HAL_TIM_PWM_Stop(htim, channel);
}
```

- Hàm `noTone`: Hàm này dừng tín hiệu PWM trên kênh Timer, dừng phát âm thanh.

```
void noTone(TIM_HandleTypeDef *htim,
        uint32_t channel) {
    // Tắt tín hiệu PWM
    HAL_TIM_PWM_Stop(htim, channel);
}
```

- Phát đoạn nhạc:

```
void playOpeningSound(TIM_HandleTypeDef *htim,
        uint32_t channel, const int notes[][3],
        size_t len) {
    for (int i = 0; i < len; i++) {
        int frequency = notes[i][0];
        int duration = notes[i][1]/3*2;
        int pause = notes[i][2]/5;

        if (frequency > 0) {
            // Phát tín hiệu PWM
            tone(htim, channel, frequency, duration);
        }
        // Tạm dừng
        HAL_Delay(duration + pause);
        // Tắt tín hiệu
        noTone(htim, channel);
    }
}
```

5. Kết quả

- Buzzer có thể phát ra các note nhạc chính xác từ file MIDI.
- Tạo ra các hàm âm thanh có thể tích hợp vào từng sự kiện của trò chơi.

3.4 Tích hợp

Hoạt động đọc tín hiệu từ nút bấm và phát âm thanh ra loa thực hiện đồng thời trên trên luồng hoạt động `defaultTask`. Việc đọc tín hiệu từ các nút sẽ được thực hiện với tần số 100Hz. Khi đọc được tín hiệu người chơi bấm nút, luồng `defaultTask` sẽ gửi 1 byte dữ liệu tới luồng xử lý trò chơi thông qua hàng đợi gói tin `osMessageQueue queue1`. Tương tự, tại luồng xử lý trò chơi khi trò chơi được bắt đầu, hoặc nổ bóng, hoặc khi kết thúc game sẽ gửi 1 byte dữ liệu thông qua `osMessageQueue queue2` để thông báo cho luồng `defaultTask` bật nhạc tương ứng với sự kiện của trò chơi.

CHƯƠNG 4. KẾT QUẢ

4.1 Đánh giá

Tựa game bắn trứng khủng long do nhóm xây dựng ứng dụng trên kit phát triển STM32F4 có đầy đủ các tính năng rất cơ bản so với tựa game dynomite! gốc được phát triển trên máy tính. Tựa game chạy tốt, ổn định trên STM32F4. Người dùng có thể chơi trò bắn trứng khủng long trên toàn bộ hệ thống của nhóm với trải nghiệm tương tự việc chơi game trên hệ máy cầm tay cũ GameBoy của Nintendo, với âm thanh trò chơi vô cùng sống động.

Tuy nhiên, do chỉ có các tính năng cơ bản nên trò chơi còn thiếu rất nhiều thứ thú vị nếu so sánh với tựa game gốc. Ngoài ra, phần âm thanh của hệ thống chưa có thành phần lý thuyết vững vàng để có thể sử dụng, phát ra tín hiệu âm thanh một cách hiệu quả triệt để. Thêm vào đó, phần xử lý game của hệ thống vẫn còn một số bug nhỏ, trong phần xử lý nổ bóng, có thể xuất hiện trong quá trình trải nghiệm của người chơi. Còn một điểm hạn chế nữa của hệ thống là phần tích hợp (mục 3.4) chỉ được xử lý trên một luồng hoạt động duy nhất, khiến cho hệ thống kém linh hoạt trong việc xử lý nhiều tác vụ tích hợp nếu mở rộng thêm.

4.2 Hướng phát triển

Dựa trên các đánh giá phía trên, ta có thể vạch ra nhiều hướng phát triển khác nhau cho hệ thống. Về tính năng của trò chơi, hiện tại game khá là đơn giản do số màu được sinh ra chỉ là ba, để tăng độ khó cũng như tính đa dạng của một game giải đố, ta có thể tăng dần số lượng màu khác nhau có thể được sinh ra trong quá trình người dùng chơi game. Ngoài ra, ta có thể thêm tính năng "Hall of Fame", lưu trữ thành tích của người chơi vào bộ nhớ của kit. Về âm thanh, hệ thống cần phải xác định kỹ càng hơn về lý thuyết sinh ra các tín hiệu mới có thể thu được kết quả là những nốt nhạc bắt tai người nghe. Cuối cùng là hệ thống có thể mở rộng đưa việc xử lý âm thanh, tín hiệu bấm nút của người chơi ra thành các luồng xử lý riêng biệt nhau để tăng tính linh hoạt.

4.3 Demo và mã nguồn

Để hỗ trợ việc tham khảo và đánh giá, nhóm đã chuẩn bị demo, báo cáo và mã nguồn của dự án ở trên link github: **https://github.com/minhphmd/EggShooting**