Arudino 3D 作品 “Maze Game”

氏名: 西滉平

2022 年 7 月 26 日制作

1　コンセプト・システムの仕様

「3D ビューによる迷路探索ゲーム “Maze Game”」を作成した. 以降, 本ゲームを「3D 迷路ゲーム」と略称で表記する. まず, 私が作成したゲームの詳細なコンセプトについて説明する.

3D 迷路ゲームのゲームコンセプト

- ゲームジャンル：迷路探索ゲーム.
- クリア条件：迷路を探索し, すべてのアイテムを集める. アイテムは迷路上に点在し, アイテムに接近することでこれらを獲得できる.
- 操作方法：ジョイスティックとコリジョンセンサー, 及び基板 (ブレッドボード) の傾きにより操作を行う.

  プレイヤーの移動

  ジョイスティックを**前後左右に倒す**ことにより, プレイヤーがスティックの動きに対応して移動する. (例) ジョイスティックを右斜め前に倒すと, プレイヤーも右斜め方向に前進する.

  3D ビューとメニュー画面の切り替え

  ジョイスティックの**スイッチを押し込む**ことでメニュー画面が表示される. スイッチを解放すると 3D ビューの画面に戻る.

  3D ビューの視点変更

  コリジョンセンサーを**押しながら**, **基板**を**基板の縦方向を軸に左右へ傾ける**と, 傾けた角度に応じて 3D ビューの視点が回転する. (例) コリジョンセンサーを押しながら基板を左手前へ 30 度傾けると, 3D ビューの視点が左方向へ 30 度回転する.

  以降, ジョイスティックのスイッチやコリジョンセンサーを単にボタンとも呼称する.
- スクリーン画面：ゲームの進行などに応じて基板上の OLED 液晶ディスプレイに各画面が表示される.

  タイトル画面

  ゲームタイトルを表示し, プレイヤーにボタンの入力を求め, ゲーム開始を促す. (図 1)

  3D ビュー

  ゲーム中のプレイヤーの位置・視点から見える視界を表示する. (図 2,3) これにより, 視界に映る迷路の壁やアイテムとの位置関係が分かる. このとき, プレイヤー操作による位置の移動や視点の変更に応じて, 視界もリアルタイムに変化する.

  メニュー画面

  ゲーム中でのプレイヤーの向きや周囲の状況が俯瞰図で表示される. 更に, 現時点での獲得アイテム数とクリアに必要な総アイテム数が表示される. (図 4)

  クリア画面

  クリアした旨を表示し, プレイヤーにボタンの入力を求め, タイトル画面への移行を促す. (図 5)

3D 迷路ゲームのゲームコンセプト (続き)

- 効果音：タイトル画面や画面切り替え時, 移動中, アイテム獲得時及びクリア時に, それぞれに対応した効果音 (SE) を発生させる.
- ゲームの進行：“タイトル画面→ゲームプレイ→クリア画面”の流れを繰り返す. ゲームプレイ中は, 3D ビューとメニュー画面が切り替え可能である.

図 1　タイトル画面

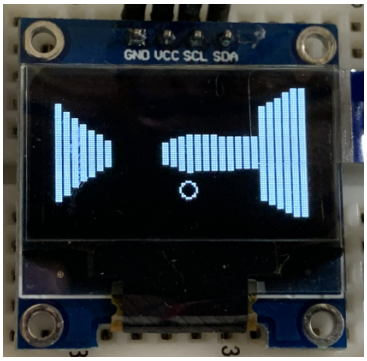

図 2　3D ビューの例 (1)

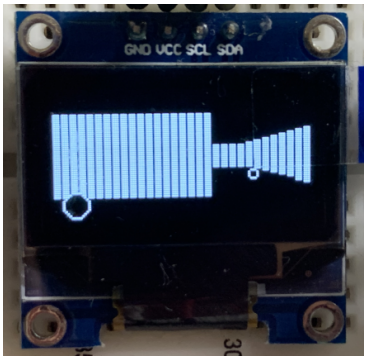

図 3　3D ビューの例 (2)

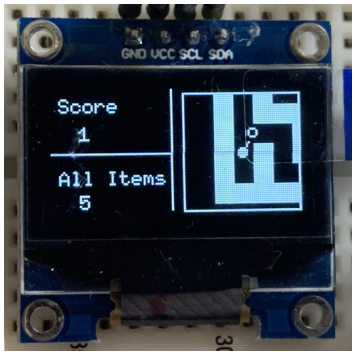

図 4　メニュー画面

図 5　クリア画面

続いて, 私が作成したゲームシステムの仕様を以下で説明する.

3D 迷路ゲームシステムの仕様

- 初期状態：本システムの初期状態は, ゲームの開始を待機する**待機状態**とする.
- 状態：本システムは, ゲームの進行により, 以下の 3 つの状態を持つ.

  待機状態

  この状態ではタイトル画面を表示し, ゲームの開始を待機する. ゲームの開始はいずれかのボタンの押下を検知して判定する.

  プレイ状態

  この状態ではゲームを実行し, プレイヤーの操作に従い, 移動や視点変更, アイテム獲得, メニュー画面への切り替えなどを行う.

  クリア状態

  この状態ではタイトル画面への移行を待機する. タイトルへの移行はいずれかのボタンの押下を検知して判定する.
- 入力：コンセプト通り, ジョイスティックとコリジョンセンサー, 基板の傾きを入力として受け取る.
- 出力：本システムからの出力として, **効果音**と**画面表示**がある. 効果音の出力は MP3 プレイヤー (YX5200) を経由してスピーカーから, 画面表示は OLED 液晶 (SSD1306) から行われる.

2　ハードウェアの回路

まず, 使用したモジュールを以下の表 1 に示す. 以降, 各モジュールは回路図上ではチップ名で表記し, レポート内では品目名で呼称する.

表 1　使用したモジュール一覧

品目	チップ名	モジュールの詳細
OLED 液晶	SSD1306	0.96 インチ, 4 ピン, 白色, IIC (I2C) 方式
MP3 プレーヤー	YX5200	microSD カードに記録されている MP3 ファイルを再生可能
6 軸センサー (IMU)	MPU6050	3 軸方向の加速度及び角速度が取得可能, I2C 方式
ジョイスティック	KS0008	方向入力が可能, スティックを押し込んでプッシュボタンとしても利用可能
コリジョンセンサー	KS0021	衝突判定が可能, 本ゲームではトリガーボタンとして利用

続いて, 以下の図 6 に作成したゲーム機体のハードウェア回路を示す.

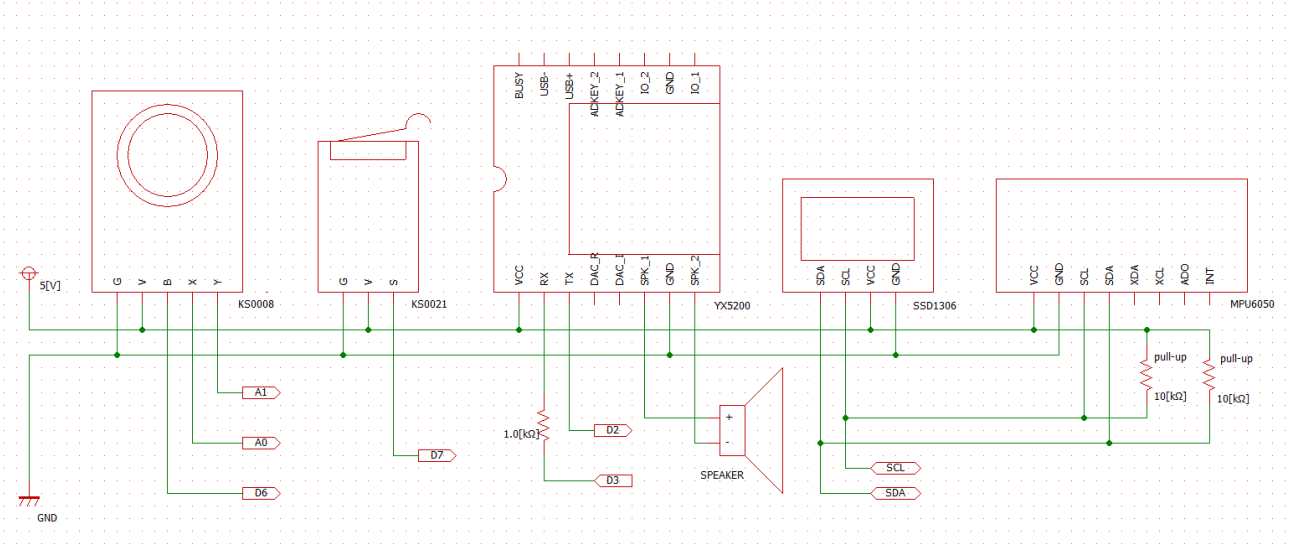

図 6　作成したゲーム機体の回路図

2.1　回路の説明

初めに, 回路の配線や抵抗の効果・意味について説明する.

まず, MP3 プレイヤー (YX5200) のシリアルポートである **RX ピン**及び **TX ピン**の電圧は **3.3[V]** である一方, Arduino 側の出力電圧は **5[V]** であるため, **1[kΩ]** の抵抗を介している. [1] これにより, Arduino 側からの送信内容を正確に MP3 プレイヤーの RX ピンで受け取ることが可能となる. MP3 プレイヤーの TX ピンからの送信内容については, 抵抗を介さずとも Arduino 側で正しく受信できるため, 抵抗は不要となっている.

次に, I2C 方式の配線について説明する. 今回使用した中で I2C 方式通信を採用しているモジュールは, **OLED 液晶 (SSD1306)** と **6 軸センサー (MPU6050)** である. この方式では, マスターデバイスである Arduino 本体とスレイブデバイスである各モジュールを **SCL** と **SDA** の 2 つのバス配線で接続する. マスターが SCL バスを通じてクロック信号をスレイブ側へと送信し, これを基に SDA バスを通じてマスター・スレイブ間でデータを送受信することが出来る. SCL バス及び SDA バスはプルアップされる必要があるが, Arduino の I2C 方式通信に必要な Wire.h ライブラリを使用すると自動的に内部プルアップ抵抗が使用されるため, 必ずしも外部プルアップ抵抗は必要ではない. 今回は通信の安定性を高めるため, 外部プルアップ抵抗として **10[kΩ]** の抵抗を電源電圧 5[V] との間に介している.

また, 本来スイッチにはプルアップ抵抗が必要となるが, ジョイスティックのスイッチ (KS0008) やコリジョンセンサー (KS0021) には既に内部に抵抗が組み込まれているため外部プルアップ抵抗が必要ではない.

2.2 取得・出力できる情報とその形式の説明

続いて, 各モジュールから取得・出力できる情報及びその形式や方法について説明する.

まず, ジョイスティックは V ピンに電源電圧 5[V], G ピンに GND を接続することで, ティックの傾けている方向及びボタンの状態が他の 3 つのピンから取得できる. X ピン及び Y ピンは, スティックの傾いている X 軸方向・Y 軸方向の成分を 0 から 1024 までのアナログ信号で出力する. **512** の値がスティック中央 (傾けていない場合) の時の信号値であり, X 軸・Y 軸について正の方向に倒していくと **512 から 1024 まで**, 負の方向に倒していくと **512 から 0 まで**信号値が変化する. 更に, B ピンはボタンが押下されているならば **"1"**, 解放されていれば **"0"** を出力する. X ピン及び Y ピンは Arduino 本体のアナログピンである A0 ピン・A1 ピンに接続し, B ピンはデジタルピンである D6 ピンに接続し, これらのピンからジョイスティックの情報を取得する.

一方で, コリジョンセンサーは V ピンに電源電圧, G ピンに GND を接続することで, S ピンからセンサーのトリガーボタンの押下が検知できるが, ジョイスティックのプッシュボタンの論理とは逆になっている. すなわち, S ピンはトリガーボタンの押下で **"0"**, 解放で **"1"** を出力する. S ピンを D7 ピンに接続し, このデジタルピンから Arudino 本体はコリジョンセンサーの状態を取得できる.

次に, MP3 プレイヤーは VCC ピンに電源電圧, GND ピンに GND を接続し, RX ピン・TX ピンを Arduino 本体のデジタルピン D3 ピン・D2 ピンと繋ぐことで, シリアル通信により Arudino 側で MP3 プレイヤーを制御でき, セットされている microSD カードの MP3 データを再生することができる. スピーカーの + 端子及び-端子は, MP3 プレーヤーの SPK_1 ピン・SPK_2 ピンに接続することで音を表すアナログ信号が + 端子へと出力され, スピーカーから MP3 データが音として出力される. シリアル通信における制御には **Paul Stoffregen 氏の SoftwareSerial.h ライブラリ**を, MP3 プレイヤーの制御には **DFRobot 社の DFRobotDFPlayerMini.h ライブラリ**を使用した.

続いて, OLED 液晶は VCC ピンに電源電圧, GND ピンに GND を接続し, SCL ピン及び SDA ピンを Arduino 本体の SCL ピン及び SDA ピンと繋がっている SCL バス・SDA バスと接続している. これにより, I2C 方式の通信が行え, Arduino 本体から OLED 液晶を制御して画面の表示を行うことができる. レジスタの読み込みや書き込み等の通信処理には **oliver 氏の U8glib.h ライブラリ**を使用した.

同様に, 6 軸センサーも VCC ピンに電源電圧, GND ピンに GND を接続し, SCL ピン及び SDA ピンを Arduino 本体の SCL ピン及び SDA ピンと繋がっている SCL バス・SDA バスと接続している. 6 軸センサー (MPU6050) には DMP と呼ばれるセンサーの姿勢角の情報を算出する内部機能があり, DMP の IC 回路によって, 誤差の少ない角度情報を取得することが可能となっている. 今回は, DMP を用いて, 姿勢角の情報に対応するクォータニオンを取得し処理を行った. I2C 方式通信による DMP の有効設定や 6 軸センサーからのクォータニオンの取得処理, 及びクォータニオンの計算処理には **Electronic Cats 社の MPU6050_6Axis_MotionApps20.h** ライブラリを使用した.

2.3 各モジュールの配置及びゲーム機体のデザイン

最後に, 基板上での各モジュールの配置 (ゲーム機体のデザイン) について説明する.

まず, 各モジュールの配置を図 7 に示す. 右上にコリジョンセンサー, 中央に OLED 液晶, その左隣に 6 軸センサー, 左上にジョイスティックが配置されている. また, 右下には Arduino 本体, その左隣に MP3 プレイヤー, 左端にスピーカーが固定されている.

基板 (ブレッドボード) をゲーム機体と見なした場合, 重視すべき点として **"配線やモジュールが固定され, 安定しているか"** と **"手に持った時の操作が容易であるか"** の 2 点を私は考えた.

まず, 前者について説明する. プレイヤーがスティックやボタンを操作し, 基板を傾ける際に, 各モジュールや配線が固定されていないと操作がしづらいことが考えられる. また, モジュール・配線が不安定であると, 基板から部品が落下する, 部品の接触が悪くなりゲームにノイズが発生する, または進行不能となる等の様々な悪影響が考えられる. これらを回避するため, 図 8 のようにモジュールのピンを基板へしっかり差し込むだけでなく, モジュールや配線自体

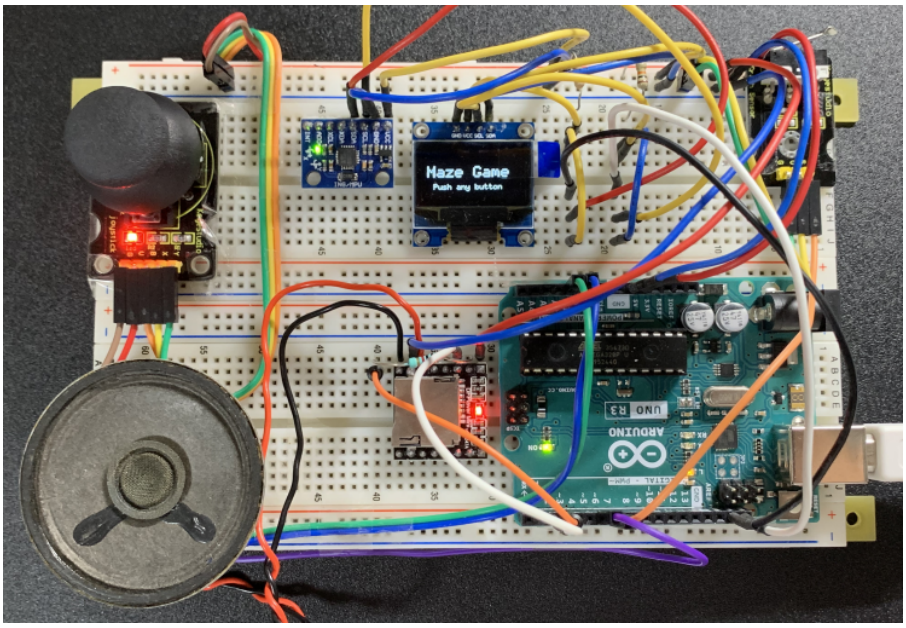

図 7　モジュールの配置及びゲーム機体のデザイン

を基板にテープで固定し, 安定性を高めた.

次に, 後者について説明する. 基板 (ゲーム機体) を手にしたとき, 指が置かれる位置に操作用のモジュールがないと, 操作性が悪いことが考えられる. 指が置かれる位置としては, 基板の右上の端・左上の端とそれらの基板の縁が挙げられる. ここで, 携帯ゲーム機の大半がジョイスティックを左に, トリガーボタンを機体の縁に設置していることを参考に, 図 7 で示したようにジョイスティックを左上の端に, コリジョンセンサーを基板右上の縁に設置することにより, 図 9 のようにプレイヤーが操作しやすい基板のデザインにすることが出来た.

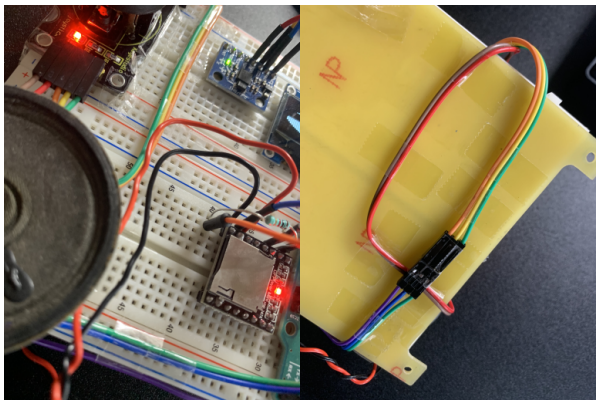

図 8　配線及びモジュールの固定

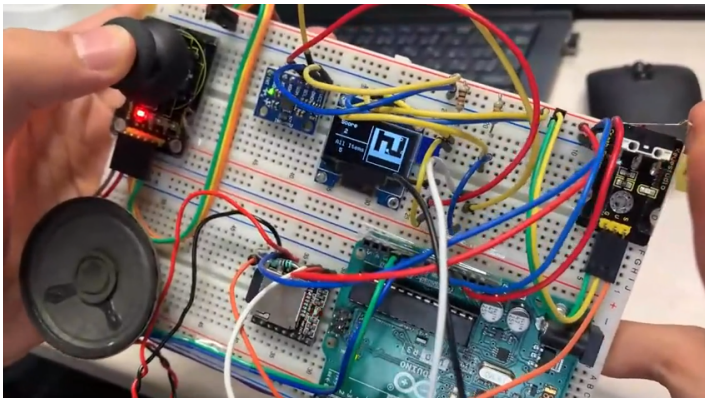

図 9　プレイ中の様子

終わりに, 基板に配置した際のジョイスティック及び 6 軸センサーの軸の向きを以下の表 2 にまとめる.

表 2　モジュールの軸と基板上での方向の対応

品目	方向軸	基板上での方向
ジョイスティック	**X 軸正**の方向	基板の**水平右方向**
	Y 軸正の方向	基板の**水平上方向**
6 軸センサー (IMU)	**X 軸正**の方向	基板の**水平下方向**
	Y 軸正の方向	基板の**水平右方向**
	Z 軸正の方向	基板の**垂直上方向**

3　ゲームプログラムの設計方法

本節では, 作成したゲームプログラムのアルゴリズムを説明する.

まず, プログラムがゲーム進行の制御処理を行う前のセットアップ処理について説明する. ゲームプログラムは以下に示す処理を起動直後に 1 度だけ行う.

MP3 プレイヤー　　**シリアル通信の開始**

- SoftwareSerial クラスのインスタンスを生成して RX・TX ピンに D2 ピン・D3 ピンを指定し, 通信速度を 9600bps に設定.
- 生成したインスタンスを DFRobotDFPlayerMini.h ライブラリの DFRobotDFPlayerMini クラスのインスタンスに渡し, MP3 プレイヤーの制御処理を開始する.

OLED 液晶　　**I2C 通信の開始及び各種表示設定**

- U8GLIB_SSD1306_128X64 クラスのインスタンスを生成し, I2C 方式による OLED 液晶との通信を開始する.
- 生成したインスタンスのメソッドを利用し, フォントの大きさや形式, 描画する色等を設定する.

6 軸センサー　　**I2C 通信の開始及び各種表示設定**

- MPU6050 クラスのインスタンスを生成し, I2C 方式による 6 軸センサーとの通信を開始する.
- 生成したインスタンスのメソッドを利用し, DMP の有効化を行う. 成功した場合は姿勢角の算出時に生じるバイアス (誤差) を較正する頻度を指定する.
- ライブラリのサンプルスケッチ “IMU_Zero” の実行結果より得た, 3 軸加速度・ジャイロセンサーのオフセット値を指定してセンサーを校正する. [2]

その他　　**単位ベクトルの設定**

- X 軸とベクトルの成す角が 0 度から 3 度刻みで 45 度までとなる, 計 $45/3+1=\mathbf{16}$ 個の単位ベクトルを求める. それぞれの角の正弦 (sin) ・余弦 (cos) の組を順に求め, 本プログラム独自のクラスである Vector2D クラス配列にこれらの組を格納する.

ここで, 16 個の単位ベクトルについて捕捉する. 0 度から 45 度までとしているのは, 三角関数の性質からこれらの正弦・余弦の組の値だけで 360 度全ての角の正弦・余弦が求まるからである. 具体的には以下のような理由である.

まず, 45 度から 90 度までの正弦・余弦の値については, $sin\theta = cos(90^\circ - \theta)$, $cos\theta = sin(90^\circ - \theta)$ より, 0 度から 45 度までの正弦・余弦の値を用いて求められる. 次に, 90 度から 180 度までの正弦・余弦の値については, $sin\theta = sin(180^\circ - \theta)$, $cos\theta = -cos(180^\circ - \theta)$ より, 0 度から 90 度までの正弦・余弦の値を用いて求められる. 最後に, 180 度から 360 度までの正弦・余弦の値については, $sin\theta = -sin(360^\circ - \theta)$, $cos\theta = cos(360^\circ - \theta)$ より, 0 度から 180 度までの正弦・余弦の値を用いて求められる. 従って, 0 度から 45 度までの正弦・余弦の値の組があれば, 0 度から 360 度までの全方向の角について正弦・余弦の値が求まる.

また, 今回のゲームシステムの実装では, 視点変更並びにレイキャスティング (後述) における光線の間隔が 3 度刻みであるため, 0 度から 3 度刻みで 45 度までの 16 個の正弦・余弦の組だけでよい. これらの値は, 単に正弦・余弦の値として使う, レイキャスティングの光線の処理に利用する等, 本プログラム内で多種多様な用途で使用される.

以上のセットアップ処理を行った後, **ゲームのリセット処理**を行う. 具体的には, ゲームの進行状態の初期化や迷路の構成, プレイヤーの位置・視点の初期化, アイテムの配置等である. リセット処理の後, ゲームの進行処理へと移る. これについては次で説明する

3.1　ゲームの進行処理

ゲームの進行状態は, 以下の図 10 のように, 3 つに分類される.

待機状態 S_0 では, MP3 プレイヤーと OLED 液晶と制御し, タイトル画面で流す SE をループさせ, タイトルとボ

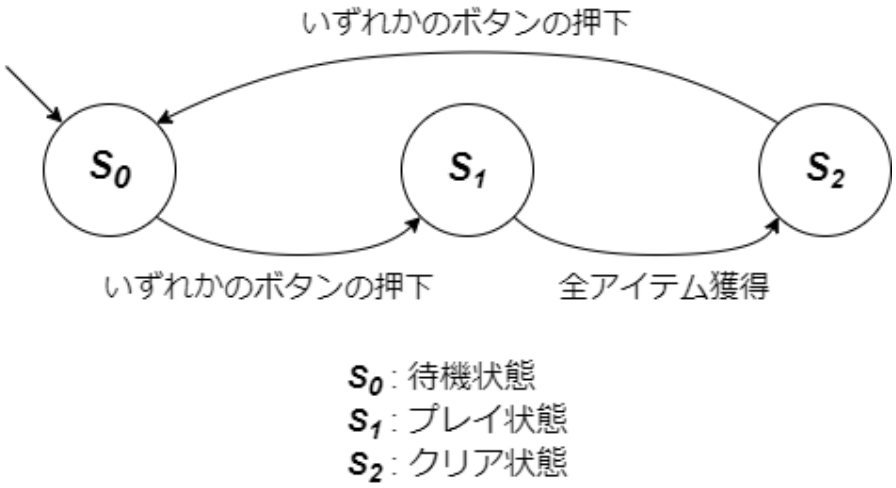

図 10　ゲームの進行に関する状態遷移

タンの入力を促す旨を画面に出力する．SE にはフリーの効果音 [3] の演出・アニメ用素材である “拍子木 1” を選んだ．ボタンの押下を検知した後, SE のループを終了し, プレイ状態 S_1 へ移行する．

次に, プレイ状態 S_1 では, 3D 迷路ゲームのプレイ中の処理を行う．プレイヤーの情報を格納している Player クラスのインスタンスを参照し, 獲得アイテムの数が総アイテム数と等しくなった場合, クリア状態 S_2 へと移行する．3D 迷路ゲームプレイ中の処理の詳細は後で詳しく説明する．

最後に, クリア状態 S_2 では, MP3 プレイヤーと OLED 液晶と制御し, クリア時に流す SE を再生し, クリアとボタンの入力を促す旨を画面に出力する．SE にはフリーの効果音 [3] の演出・アニメ用素材である “レベルアップ” を選んだ．ボタンの押下を検知した後, ゲームのリセット処理を行い, 待機状態 S_0 へ移行する．

3.2　プレイ状態における処理

続いて, プレイ状態における処理は, 図 11 のように, 2 つの状態を用いて表現できる．状態 Q_0 は 3D ビューの描画処理, Q_1 はメニュー画面の描画処理と対応している．

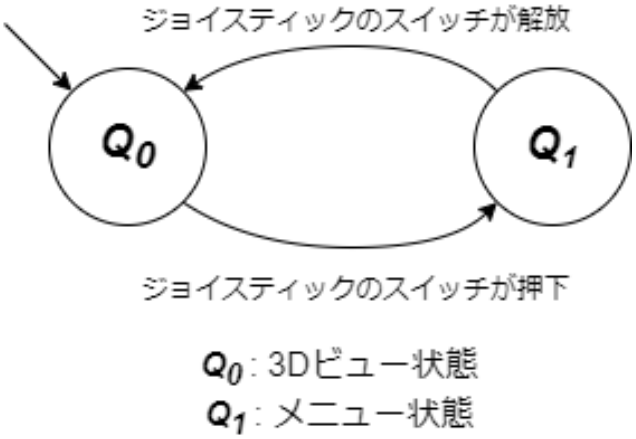

図 11　プレイ状態 $\boldsymbol{S_1}$ における内部の状態遷移

まず, 両状態で共通の処理に **“プレイヤーの位置移動”** と **“アイテムの獲得”**, **“プレイヤーの視点変更”** がある．

位置移動については, ジョイスティックの入力をプレイヤーの視点角 (視野の中心角) を基に, マップ上の X 成分・Y 成分に分割し, プレイヤーの座標にこれらを加算する．移動する場合, すなわちマップ上の X 成分・Y 成分の値が大きい場合は, フリーの効果音 [3] の生活カテゴリ内の “アスファルトの上を歩く 1” の SE をループ再生させ, 歩いて

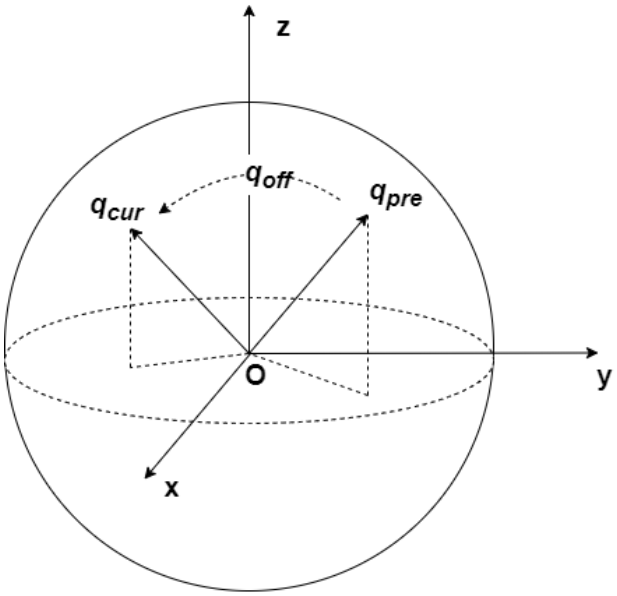

図 12　視点変更におけるオフセット角を表すクォータニオンの求め方

いる臨場感が出るようにした.

アイテムの獲得では, 未獲得のアイテムとの距離を算出し, この距離が基準未満であるものを獲得する. アイテムを獲得すると, アイテムの獲得情報を更新し, 現獲得アイテムが総アイテム数と等しい場合は状態 S_2 へ移行する. また, アイテムの獲得時には, SE としてフリーの効果音 [3] の演出・アニメ用素材である “ピアノの単音” を流す.

視点変更については, 6 軸センサーから取得できる**クォータニオン**を用いて処理を行った. 具体的な処理手順は以下の通りである.

手順 (1) コリジョンセンサーが押下されていない場合, 6 軸センサーから最新の姿勢角を表すクォータニオンを取得し, クォータニオンの変数 $\boldsymbol{q_{pre}}$ を最新のクォータニオンに更新する. また, 現在の視点角を保持しておく.

手順 (2) コリジョンセンサーが押下されている間, 6 軸センサーから現在のクォータニオンを取得し, これを $\boldsymbol{q_{cur}}$ に格納する. この $\boldsymbol{q_{cur}}$ と $\boldsymbol{q_{pre}}$ から視点変更のオフセット角を表すクォータニオン $\boldsymbol{q_{off}}$ を取得し, $\boldsymbol{q_{off}}$ を用いて Y 軸についての回転角を得る. 求めた回転角と保持している視点角を加算したものを現在の視点角として更新する. 但し, このときの度数法による角度の値は 3 の倍数 (3 度刻み) となるように, 値を丸める.

手順 (3) コリジョンセンサーが解放された場合, 現在の視点角を確定させ, 視点変更の処理を終了する.

オフセット角を表すクォータニオン $\boldsymbol{q_{off}}$ の求め方

図 12 に示す通り, $\boldsymbol{q_{cur}}, \boldsymbol{q_{pre}}, \boldsymbol{q_{off}}$ について, 以下の等式が成立する.

$$q_{cur} = q_{off} \otimes q_{pre}$$

ここで, q_{pre} は単位クォータニオンであり, これの共役 q_{pre}^* は q_{pre} の逆クォータニオンである. 従って,

$$q_{cur} \otimes q_{pre}^* = q_{off} \otimes q_{pre} \otimes q_{pre}^* = q_{off} \otimes (q_{pre} \otimes q_{pre}^{-1}) = q_{off} \qquad \therefore \boldsymbol{q_{off} = q_{cur} \otimes q_{pre}^*}$$

以上より, 最後に求めた式より, $\boldsymbol{q_{off}}$ が求まる. クォータニオンの計算並びにクォータニオンが表すオフセット角 (姿勢角) の算出には 6 軸センサーのライブラリ内で実装されているものを利用した.

次に, 状態 Q_0 では, **レイキャスティング** [4] を用いて, 二次元のプレイヤー座標や迷路マップの情報から 3D ビューの画面を描画する. レイキャスティングは以下の手順で実行される. 以下, 視点角 (向いている方向) を α, 視野角を β, 光線の間隔の角を ω とする.

手順 (1) 図 13, 14 の左に示されるように, プレイヤーの位置を始点として, 視点角 α から左右に $\frac{\beta}{2}$, 全てで β の範囲

を ω 毎に分割する．範囲の端と分割時に引いた線をプレイヤーの座標から発射された光線とみなす．光線数 n は, $n = \beta/\omega + 1$ であり, 各光線 $l_i (1 \leq i \leq n,$ 右端が $i = 1)$, 視点角 α と光線 l_i が成す角を θ_i とする．

手順 (2) 発射した各光線 l_i について, 迷路の壁との交点を求め, 最も初めに衝突した交点とプレイヤーとの距離 d_i を算出する．プレイヤーから遠くに存在する壁は遠近法の考えでは 3D ビュー上で見えなくなるため, d_i が大きい場合は, 壁との交点を発見出来なかったと判断する．

手順 (3) 各光線 l_i について求めた距離 d_i を用いて, 描画に用いる壁の高さを決定する．遠くのものほど小さく映ればよいため, 壁の高さは d_i に反比例させればよい．但し, 算出した距離をそのまま用いると図 13 のように歪みが発生するため, 各光線 l_i と視点角 α が成す角である θ_i の余弦の値と d_i の積を取った値に反比例するように壁の高さを決定する．交点が発見できなかった場合は, $h_i = 0$ (壁が見えない) と設定する．

$$\text{壁の高さ } h_i = \frac{\text{定数}}{d_i \times cos\theta_i}$$

手順 (4) 図 14 の右に示されるように, 壁の高さ h_i に基づき, 右端から左端まで画面に描画する．そのあとに, 未獲得のアイテムとの距離及び視点角 α との成す角を求め, このアイテムが視界内かつプレイヤーとアイテムの間に壁が存在しないことを先の手順で求めた d_i などの情報を用いて決定し, 壁を描画した上からアイテムを描画する．アイテムの大きさもプレイヤーとの距離に反比例させて, 遠いものほど小さく映るようにする．

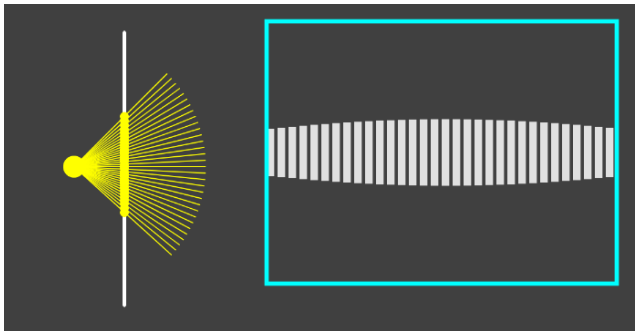

図 13　前面に壁がある場合の歪み

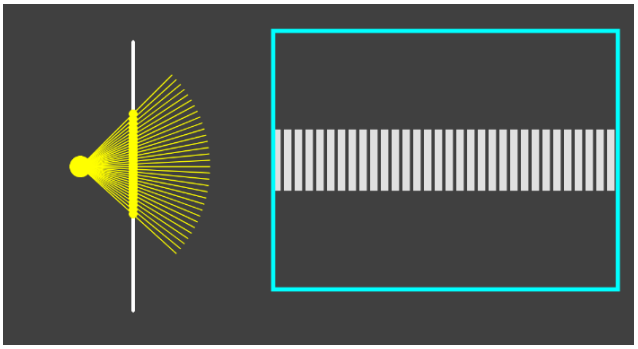

図 14　歪み補正した後の 3D ビュー

以上の手順により, 図 2 及び 3 のような 3D ビューの画面が描画できる．

一方で, 状態 Q_1 では, メニュー画面を表示する．図 4 で示した通り, メニュー画面の右側には現獲得アイテム数とクリアに必要な総アイテム数を表示し, 左側にはプレイヤーを中心として周囲の迷路の様子を表した俯瞰図を表示させている．俯瞰図の描画は, プレイヤーの位置情報から対応する迷路の二次元データの添字の組を獲得し, これを利用して, 壁もしくは道かを 1 マスずつ判断し塗りつぶしている．その後, プレイヤーの視点角や未獲得アイテムの所在位置情報を取得し, これらを基にプレイヤー・アイテムをマップに描画する．

以上の 2 状態 Q_0, Q_1 を, ジョイスティックのプッシュボタンの押下・解放によって切り替えることで, 3D 迷路ゲームにおける 3D ビューとメニュー画面の切り替えを可能にした．また, 状態 Q_0 から Q_1 へ移行する際には, フリーの効果音 [3] の演出・アニメ用素材である “シーン切り替え 1” を流した．

4　考察・工夫点

本課題を通して, システムを仕様からハードウェア, 制御プログラムに至るまで全て自力で作成することを経験した．本課題のようなシステムを全て一から考えて作り上げる経験は, 今回が初めてであったので, システムの仕様の重要性を身をもって実感することが出来た．仕様の整合性がとれていてある程度細かく定まっていないと, ハードウェア・制御プログラムの開発において開発内容にずれや不具合が生じて開発に支障をきたすため, 十分留意して仕様を決定する必要があると学んだ．特に, 私が作成したような規模の大きなシステムとなると, 開発における仕様の重要性は顕著であったと考えられる．

また, 今回作成した “3D 迷路ゲーム” は **“処理速度”**, **“メモリ量”**, **“デザイン”** の 3 つの点において, 工夫を有し

ていると考察できる. それぞれにおける具体的な工夫の例や詳細は, 以下に示す.

工夫点の詳細

処理速度

単位ベクトルを予め求めておくことにより, レイキャスティングにおける各光線の角の正弦・余弦の値を高速に求めることが出来る. sin 関数及び cos 関数によって正弦・余弦を求める際は, 高い計算コストを要するが, 予め求めた値にアクセスする場合は関数を用いて計算する時と比較して非常に高速に処理が行える. 他にも, プレイヤーの移動やアイテムとの位置関係の把握などの処理でも正弦・余弦の値が必要となるため, 正弦・余弦の値を用いる処理は多種多様な用途で頻繁に発生することが分かる. 従って, 単位ベクトルを予め求めることで正弦・余弦の算出一回が高速に行えるため, 全体の処理速度が向上する.

メモリ量

メモリ量の節約は, 予め算出している単位ベクトルの個数に特に表れている. 三角関数の性質を用いることで, 本来 $360/3 = \mathbf{120}$ **個**必要となる単位ベクトルを **16 個 (約** $\frac{\mathbf{1}}{\mathbf{8}}$**)** まで削減することが出来た. また, OLED 液晶に出力するためのライブラリの選出にも気を配り, 他のライブラリだとフラッシュメモリやメインメモリに書き込まれるメモリ量が多く動作しないことがあったため, “U8glib.h” ライブラリを使用した. 以上の工夫によって以下のように正常に動作できるだけのメインメモリを確保することが出来た.

デザイン

ゲーム機体やゲームデザインからも分かるように, プレイしやすさ・分かりやすさを考えて, ゲームを作成した. ゲーム機体については, 携帯ゲーム機 (任天堂の 3DS 等) を参考に, トリガーボタンを基板の縁寄りに配置し, ジョイスティックを基板左側に配置し, 更に画面が見やすいように OLED 液晶は中央に配置した. また, 3DS から着想を得て, 6 軸センサーを用いて基板の傾きから視点を移動させることを思いついた. このような工夫により, プレイヤーが初めてでも直感的にプレイしやすいゲーム機体を設計できたと考えられる. 一方で, ゲームデザインは, メニュー画面でのアイテム数や俯瞰図の情報によって, プレイヤーが置かれている状況が逐次確認できるため, 分かりやすいものとなっていると考えられる. 更に, 各場面や動作に応じて流れる SE もゲームへの没入感を高める効果があり, ゲームの完成度をより高める要因となっていると考察できる.

5 感想

本課題では, 実際にゲーム開発を経験し, ゲームを制作する大変さを身に染みて理解することが出来た. 考察でも述べたように, 仕様・ハードウェア・制御プログラムの一連の開発で, それぞれの開発内容でかみ合わなくなる箇所が発生する等, 一からゲーム開発することの難しさを体感できた. また, バグや予期せぬ挙動も多数発生したため, これらを取り除くのに非常に苦労した.

また, Youtube の動画や普段の授業・勉強の中から学んだ内容であるレイキャスティング [4] やクォータニオンといった概念やテクニックを利用することで, 実際のプログラム作成に活用することが出来た. 自分の知識を実践し, 生きた知識とすることが出来て, 良い経験となったと感じた. 今後も, 自分の知識を実践する機会を積極的に活用し, 血となり肉となるように努めていきたい.

6 使用素材・参考文献

[1] DFRobot WIKI EN, DFR0299 DFPlayer Mini (https://wiki.dfrobot.com/DFPlayer_Mini_SKU_DFR0299)

[2] MPU6050 の DMP を使う, Qiita, Ninagawa_Izumi 氏, (https://qiita.com/Ninagawa_Izumi/items/2646c7d3d98943919f80)

[3] 効果音ラボ (https://soundeffect-lab.info/)

[4] 高校数学と JavaScript だけ。FPS の作り方 #1 〜 #3, ヘロンの数学チャンネル (https://t.co/jgoROJ14f5)