4. IMPLEMENTASI

Pada bab ini akan dijelaskan implementasi kode dari bab sebelumnya.

Tabel 4.1 Komponen yang Diimplementasi

Algoritma	Keterangan
Prioritas Antarmuka Game	Tampilan Game
Alur Game	Sistem Game dari awal hingga akhir
Pergerakan Pemain	Manipulasi posisi pemain

4.1 Antarmuka Game

4.1.1 Implementasi Peta

Segmen 4.1 Kode Generasi Peta

```
// Map Tiles Info Array
  this.registry.set('mapInfo', []);
  let mi = this.registry.get('mapInfo');
  for (let i = 0; i <= 5; i++) {
      // Branch Tiles Info Array
      let ar = [];
      let maxDebuff = 3; // Maximum Debuff Tiles Per Branch
      let maxEvent = 3; // Maximum Event Tiles Per Branch
      if (i === 1) {
          maxDebuff += 2;
          maxEvent -= 2;
      } else if (i === 3) {
          maxDebuff += 1;
          maxEvent -= 1;
      }
      for (let j = 0; j <= 2; j++) {
          for (let k = 0; k <= 4; k++) {
              let rand = Phaser.Math.Between(0,9);
              // Tile Possibility Array
```

```
// 0 = Menu Tile, 1 = Debuff Tile,
// 2 = Unique Event Tile
if (rand <= 4) {
    ar.push(0);
} else if (rand <= 7) {
    if (maxDebuff > 0) {
        ar.push(1);
        maxDebuff--;
    } else {
        let rand2 = Phaser.Math.Between(0,1);
        if (rand2 === 0) {
            ar.push(0);
        } else {
            if (maxEvent > 0) {
                ar.push(2);
                maxEvent--;
            } else {
                ar.push(0);
            }
        }
    }
} else {
    if (maxEvent > 0) {
        ar.push(2);
        maxEvent--;
    } else {
        let rand2 = Phaser.Math.Between(0,1);
        if (rand2 === 0) {
            ar.push(0);
        } else {
            if (maxDebuff > 0) {
                ar.push(1);
                maxDebuff--;
            } else {
                ar.push(0);
            }
```

```
                            }
                        }
                    }
                }
                // 3 = Money Tile
                if ((i === 2 && (j === 0 || j === 1)) || (i === 4
&& j === 2) || (i === 5 && j === 1)) {
                    ar.push(0);
                } else if ((i === 1 && j === 1) || (i === 5 && j
=== 2)) {
                    ar.push(1);
                } else if (i === 3 && (j === 0 || j === 1)) {
                    ar.push(2);
                } else {
                    ar.push(3);
                }
            }
            mi.push(ar);
        }
        this.registry.set('mapInfo', mi);
```

Peta dibuat berupa *array* dua dimensi yang mengandung 6 *array* yang kemudian berisi 18 data jenis petak yang akan digunakan dalam permainan. Petak dipilih secara acak dan kemudian peluang terpilihnya satu petak dengan yang lainnya diubah dengan *weighted random algorithm* untuk membuat peluang munculnya petak menu lebih sering dan petak *debuff* dan *event* lebih jarang, dengan rasio 0-4 untuk menu, 5-7 untuk *debuff*, dan 8 dan 9 untuk *event*, untuk memfokuskan pemain dalam membuat keputusan seputar menu yang dipilih. Jumlah petak *debuff* dan *event* yang dapat muncul dalam satu cabang kemudian dibatasi lagi dengan sebuah jumlah maksimum sesuai dengan implikasi cabangnya di dunia nyata, misal cabang makan diluar akan mendapat lebih banyak *debuff* dan petak uang ada yang diganti dengan petak *debuff* karena makan diluar lebih mahal dan kurang sehat daripada memasak makanan sendiri di rumah, cabang makanan camilan akan memiliki lebih banyak *event* karena adanya waktu luang, dan seterusnya.

4.1.2 Rendering Peta

Segmen 4.2 Kode Rendering Peta

```
      // Game Scene Display

       this.bg        =        this.add.image(512,        384,
'background').setAlpha(0.5);
       let  b0Image  =  this.add.image(377  +  125,  655  +  50,
'Branch0').setScale(1.25);
       let  b1Image  =  this.add.image(323  +  50,  384  +  60,
'Branch1');
       let  b2Image  =  this.add.image(824  +  100,  529  +  75,
'Branch2');
       let  b3Image  =  this.add.image(740  +  40,  362  +  40,
'Branch3');
       let  b4Image  =  this.add.image(520  +  40,  312  +  40,
'Branch4');
       let  b5Image  =  this.add.image(579  +  40,  127  +  40,
'Branch5');

      // Map Tiles Display
       // Tile Coordinates
       // Branch 0 Array
       let b0a = [[196, 612], [223, 619], [253, 615], [278, 630],
[308, 625], [338, 624],
               [365, 619], [395, 619], [425, 625], [455, 630],
[482, 618], [512, 619],
               [539, 609], [568, 615], [595, 604], [619, 619],
[645, 604], [674, 611]];
       // Branch 1 Array
       let b1a = [[177, 544], [203, 529], [233, 524], [263, 519],
[293, 524], [323, 529],
               [349, 544], [378, 539], [404, 524], [428, 539],
[458, 544], [487, 549],
               [517, 544], [546, 549], [575, 544], [601, 544],
[630, 559], [659, 564]];
```

```
        // Branch 2 Array
        let b2a = [[775, 574], [794, 551], [816, 531], [842, 516],
[869, 503], [895, 489],
                [921, 476], [941, 454], [956, 429], [965, 400],
[970, 370], [973, 340],
                [973, 310], [972, 280], [970, 250], [965, 221],
[954, 193], [938, 168]];
        // Branch 3 Array
        let b3a = [[746, 519], [756, 492], [741, 467], [721, 445],
[698, 426], [680, 402],
                [683, 372], [698, 346], [724, 331], [754, 327],
[784, 322], [814, 327],
                [844, 327], [870, 314], [893, 295], [903, 266],
[906, 235], [903, 204]];
        // Branch 4 Array
        let b4a = [[845, 198], [830, 224], [806, 243], [778, 254],
[750, 265], [725, 281],
                [697, 292], [667, 296], [637, 296], [607, 296],
[577, 292], [547, 288],
                [517, 284], [487, 281], [460, 269], [434, 254],
[411, 235], [396, 209]];
        // Branch 5 Array
        let b5a = [[821, 137], [792, 129], [762, 125], [733, 133],
[707, 148], [692, 174],
                [687, 204], [681, 234], [656, 251], [627, 259],
[597, 257], [570, 244],
                [554, 218], [537, 193], [512, 175], [484, 161],
[454, 155], [424, 159]];
        // Tile Reference Container
        // Branch 0-5 Array Info
        let b0ai = [];
        let b1ai = [];
        let b2ai = [];
        let b3ai = [];
        let b4ai = [];
        let b5ai = [];
```

```
        // Branch Display
        // Branch 1
        let b1d = this.add.polygon(0, 0,
        [[0, -50],[-43.3, -25], [-43.3, 25], [0, 50], [43.3, 25],
[43.3, -25]],
        0xd9d9d9);
        b1d.setOrigin(0,0);
        b1d.x = 138;
        b1d.y = 589;
        // Branch 2
        let b2d = this.add.polygon(0, 0,
            [[0, -50],[-43.3, -25], [-43.3, 25], [0, 50], [43.3,
25], [43.3, -25]],
            0xd9d9d9);
        b2d.setOrigin(0,0);
        b2d.x = 717;
        b2d.y = 569;
        // Branch 3
        let b3d = this.add.polygon(0, 0,
            [[0, -50],[-43.3, -25], [-43.3, 25], [0, 50], [43.3,
25], [43.3, -25]],
            0xd9d9d9);
        b3d.setOrigin(0,0);
        b3d.x = 880;
        b3d.y = 150;
        // Branch 4 / Finish Tile
        let b4d = this.add.polygon(0, 0,
            [[0, -50],[-43.3, -25], [-43.3, 25], [0, 50], [43.3,
25], [43.3, -25]],
            0xd9d9d9);
        b4d.setOrigin(0,0);
        b4d.x = 365;
        b4d.y = 160;
```

```
        // Tile Color Determinant (0 = Menu = Blue, 1 = Debuff =
Red, 2 = Event = Yellow, 3 = Money = Green)
        // and Tile Creation
        for (let j = 0; j < 6; j++) {
            for (let i = 0; i < b1a.length; i++) {
                let color = [];
                if (mi[j][i] === 0) {
                    color.push(0x00ffff);
                } else if (mi[j][i] === 1) {
                    color.push(0xff0000);
                } else if (mi[j][i] === 2) {
                    color.push(0xffff00);
                } else {
                    color.push(0x00ff00);
                }

                if (j === 0) {
                    let   tile   =   this.add.circle(b0a[i][0],
b0a[i][1], 15, color[0]);
                    b0ai.push(tile);
                } else if (j === 1) {
                    let   tile   =   this.add.circle(b1a[i][0],
b1a[i][1], 15, color[0]);
                    b1ai.push(tile);
                } else if (j === 2) {
                    let   tile   =   this.add.circle(b2a[i][0],
b2a[i][1], 15, color[0]);
                    b2ai.push(tile);
                } else if (j === 3) {
                    let   tile   =   this.add.circle(b3a[i][0],
b3a[i][1], 15, color[0]);
                    b3ai.push(tile);
                } else if (j === 4) {
                    let   tile   =   this.add.circle(b4a[i][0],
b4a[i][1], 15, color[0]);
                    b4ai.push(tile);
```

```
                } else if (j === 5) {
                    let tile = this.add.circle(b5a[i][0],
b5a[i][1], 15, color[0]);
                    b5ai.push(tile);
                }
            }
        }
        this.bai = [b0ai, b1ai, b2ai, b3ai, b4ai, b5ai];
```

Koordinat untuk semua petak di peta ditentukan secara *hardcode* kemudian di-render sesuai bentuknya, dan terakhir disimpan untuk diakses guna perpindahan pion pemain.

4.1.3 Implementasi Status Pemain

Segmen 4.3 Kode Status Pemain

```
// Player Variables Initialization
        // [Protein, Carbohydrates, Fat, Calories, Money, Current
Tile, Branch, Finished, Score]
        this.game.p1Data = [0,0,0,0,3,-1,-1,false,0];
        this.game.p2Data = [0,0,0,0,3,-1,-1,false,0];
        this.game.p3Data = [0,0,0,0,3,-1,-1,false,0];
        this.game.p4Data = [0,0,0,0,3,-1,-1,false,0];

        // Debuff State
        this.game.p1Debuff = [false, false, 0, 0];
        this.game.p2Debuff = [false, false, 0, 0];
        this.game.p3Debuff = [false, false, 0, 0];
        this.game.p4Debuff = [false, false, 0, 0];

        // Nutrition Ratio
        this.game.p1Ratio = [0, 0, 0];
        this.game.p2Ratio = [0, 0, 0];
        this.game.p3Ratio = [0, 0, 0];
        this.game.p4Ratio = [0, 0, 0];

        let pti = 0; //Player Turn Index
```

```
        console.log('Player ' + playOrder[pti] + '\'s Turn \nRoll
Dice');
        let diceRoll;
```

Segmen kode 4.3 bertujuan untuk menyimpan data pemain yang akan digunakan sepanjang permainan, dengan data pemain disimpan dalam tiga *array*. *Array* pertama memiliki format protein, karbohidrat, lemak, kalori, uang, posisi petak sekarang, posisi cabang sekarang, *boolean* apakah pemain sudah mencapai akhir atau tidak, dan skor pemain. *Array* kedua menyimpan kondisi *debuff* pemain yang berlaku selama berbagai giliran seperti *debuff* diare dan kram, dengan format apakah *debuff* aktif atau tidak dan sisa giliran aktif *debuff* tersebut. *Array* ketiga menyimpan data rasio keseimbangan gizi pemain, dalam bentuk persentase dengan urutan persentase protein, karbohidrat, dan lemak.

4.1.4 Implementasi Giliran

Segmen 4.4 Algoritma dan Fungsi Giliran

```
// Player Order Random
        let playOrder = [];
        let ord = [1,2,3,4];

        for (let i = 0; i <= 3; i++) {
            let index = Phaser.Math.Between(0, 3 - i);
            playOrder.push(ord[index]);
            ord.splice(index, 1);
        }
        this.playOrder2 = playOrder.slice();

// Player Turn Change
        let change = false;
        function turnChange (tref) {
            console.log("turnChange");
            console.log("Player " + playOrder[pti] + "\'s Turn
Finished");
            if (change) { // If a player has reached the finish
                playerMove(playOrder[pti], tref);
                change = false;
                breakpoint = false;
```

```
                playOrder.splice(pti, 1);
                tref.playOrder2.splice(pti, 1);
                if (pti === playOrder.length || pti === 0) {
                    // If playerTurnIndex is out of bounds, move
to front
                    // If playerTurnIndex is at front, don't
move
                    pti = 0;
                }
                updateTurns(playOrder, tref);
            } else if (pti === playOrder.length - 1) {
                // If playerTurnIndex is the last index, move to
front
                console.log("Front Player's Turn");
                pti = 0;
            } else {
                console.log("player : " + playOrder[pti]);
                pti++;
                console.log("player : " + playOrder[pti]);
            }
            console.log(playOrder.length);
            if (playOrder.length <= 0) { // If no players are
left, end game
                console.log("endGame");
                tref.scene.start("GameOver");
                return;
            }

            // Get Player References
            let pd;
            if (playOrder[pti] === 1) {
                pd = tref.game.p1Data.slice();
            } else if (playOrder[pti] === 2) {
                pd = tref.game.p2Data.slice();
            } else if (playOrder[pti] === 3) {
                pd = tref.game.p3Data.slice();
```

```
        } else if (playOrder[pti] === 4) {
            pd = tref.game.p4Data.slice();
        }
        console.log("changePlayers");
        console.log(playOrder[pti]);
        console.log(pd);
        if (pd[6] === -1) { // If player starts at a branch
            console.log("changeAtBranch");
            tref.dice.disableInteractive().setScale(0);
            stopBranch(playOrder[pti], 0, tref);
        } else {
            tref.dice.setInteractive().setScale(2);
            tref.dice.play("rolling");
        }
    }
```

Segmen kode 4.4 menunjukkan pengacakan giliran pemain sesuai dengan jumlah pemain yang dipilih sebelum permainan dimulai, dan kode pergantian pemain yang berisi pengecekan jika ada pemain mencapai petak akhir atau tidak dan jika sudah tidak ada lagi pemain yang bermain dan mengakhiri permainan.

4.2 Alur Game

Alur game dimulai dengan input pemain yang kemudian memicu berbagai sistem lain yang ada di dalam game. Karena input pemain termasuk pergerakan pemain, maka untuk alur game akan dijelaskan berbagai macam sistem yang dapat dimulai oleh pemain.

4.2.1 Fungsi Branch

Segmen 4.5 Kode Fungsi Branch

```
// Branch Tile Process
    function stopBranch (player, diceRoll, tref) {
        console.log("stopBranch");
        // Select Branch Before Rolling Dice
        tref.dice.disableInteractive().setScale(0);

        // Get Player References
```

```
            let pd, playerDebuffState;
            if (player === 1) {
                pd = tref.game.p1Data.slice();
                playerDebuffState = tref.game.p1Debuff;
            } else if (player === 2) {
                pd = tref.game.p2Data.slice();
                playerDebuffState = tref.game.p2Debuff;
            } else if (player === 3) {
                pd = tref.game.p3Data.slice();
                playerDebuffState = tref.game.p3Debuff;
            } else if (player === 4) {
                pd = tref.game.p4Data.slice();
                playerDebuffState = tref.game.p4Debuff;
            }

            console.log("branch bef : " + pd[6]);
            let selection = pd[6];
            let sqText  = tref.add.text(tref.game.canvas.width/2,
tref.game.canvas.width/2 - 300, "Choose Branch", {
                fontFamily: 'Arial Black', fontSize: 38, color:
'#ffffff',
                stroke: '#000000', strokeThickness: 8,
                align: 'center'
            }).setOrigin(0.5);
            let                    sqA1                    =
tref.add.rectangle(tref.game.canvas.width/2        -        200,
tref.game.canvas.height/2, 250, 250, 0xffffff);
            sqA1.setAlpha(0.5);
            let                    sqB1                    =
tref.add.rectangle(tref.game.canvas.width/2        +        200,
tref.game.canvas.height/2, 250, 250, 0xffffff);
            sqB1.setAlpha(0.5);

            // Branch Selection Choices
            let branchA, branchB;
            if (selection === -1) {
```

```
                // Branch 0 or 1
                branchA                                          =
tref.add.image(tref.game.canvas.width/2          -          200,
tref.game.canvas.height/2, 'Branch0');
                branchA.setInteractive().setDisplaySize(150,
150);
                branchB                                          =
tref.add.image(tref.game.canvas.width/2          +          200,
tref.game.canvas.height/2, 'Branch1');
                branchB.setInteractive().setDisplaySize(150,
150);
                selection = 0;
            } else if (selection === 0 || selection === 1) {
                // Branch 2 or 3
                branchA                                          =
tref.add.image(tref.game.canvas.width/2          -          200,
tref.game.canvas.height/2, 'Branch2');
                branchA.setInteractive().setDisplaySize(150,
150);
                branchB                                          =
tref.add.image(tref.game.canvas.width/2          +          200,
tref.game.canvas.height/2, 'Branch3');
                branchB.setInteractive().setDisplaySize(150,
150);
                selection = 1;
            } else if (selection === 2 || selection === 3) {
                // Branch 4 or 5
                branchA                                          =
tref.add.image(tref.game.canvas.width/2          -          200,
tref.game.canvas.height/2, 'Branch4');
                branchA.setInteractive().setDisplaySize(150,
150);
                branchB                                          =
tref.add.image(tref.game.canvas.width/2          +          200,
tref.game.canvas.height/2, 'Branch5');
```

```
                branchB.setInteractive().setDisplaySize(150,
150);
                selection = 2;
            }
            pd[5] = -1;
            console.log("pd5reset : " + (pd[5] === -1));
            if (player === 1) {
                tref.game.p1Data = pd.slice();
            } else if (player === 2) {
                tref.game.p2Data = pd.slice();
            } else if (player === 3) {
                tref.game.p3Data = pd.slice();
            } else if (player === 4) {
                tref.game.p4Data = pd.slice();
            }

            // Player Input
            tref.input.on('gameobjectdown',    function(pointer,
gameObject) {
                // Branch A
                if (gameObject === branchA) {
                    if (selection === 0) {
                        // Branch 0
                        pd[6] = 0;
                    } else if (selection === 1) {
                        // Branch 2
                        pd[6] = 2;
                    } else if (selection === 2) {
                        // Branch 4
                        pd[6] = 4;
                    }
                    console.log("branch aft : " + pd[6]);
                    console.log(pd);
                    if (player === 1) {
                        tref.game.p1Data = pd.slice();
                    } else if (player === 2) {
```

```
        tref.game.p2Data = pd.slice();
    } else if (player === 3) {
        tref.game.p3Data = pd.slice();
    } else if (player === 4) {
        tref.game.p4Data = pd.slice();
    }
    branchA.disableInteractive().setScale(0);
    branchB.disableInteractive().setScale(0);
    sqText.setScale(0);
    sqA1.setScale(0);
    sqB1.setScale(0);
    if (diceRoll === 0) {
        if (breakpoint) {
            playerMove(playOrder[pti], tref);
            turnChange(tref);
            breakpoint = false;
        }
        tref.dice.setInteractive().setScale(2);
    } else {
        if (breakpoint) {
            breakpoint = false;
        }
        let curt = pd[5];
        let curb = pd[6];
        let fin = pd[7];
        while (diceRoll > 0) {
            curt++;
            diceRoll--;
        }
        pd[5] = curt;
        console.log(pd);
        if (player === 1) {
            tref.game.p1Data = pd.slice();
        } else if (player === 2) {
            tref.game.p2Data = pd.slice();
        } else if (player === 3) {
```

```
                    tref.game.p3Data = pd.slice();
                } else if (player === 4) {
                    tref.game.p4Data = pd.slice();
                }
                // Stopped Moving / Landed at Tile
                if (diceRoll === 0 && !fin && curt <= 18)
{
                    console.log("Stopped");
                    console.log("pd[5] : " + pd[5]);
                    playerMove(playOrder[pti], tref);

                    if (playerDebuffState[2] < 0) {
                        playerDebuffState[2]--;
                        if (playerDebuffState[2] === 0) {
                            playerDebuffState[0]       =
false;
                        }
                    }
                    if (playerDebuffState[3] < 0) {
                        playerDebuffState[3]--;
                        if (playerDebuffState[3] === 0) {
                            playerDebuffState[1]       =
false;
                        }
                    }
                    // Landed at Menu Tile
                    if (mi[curb][curt] === 0) {
                        console.log("Menu Tile");

                        //  Select  Current  Branch  Menu
Catalogue
                        let choices = [];
                        let source = [];
                        if (curb === 0) {
                            source = branch0.slice();
                        } else if (curb === 1) {
```

```
                source = branch1.slice();
            } else if (curb === 2) {
                source = branch2.slice();
            } else if (curb === 3) {
                source = branch3.slice();
            } else if (curb === 4) {
                source = branch4.slice();
            } else {
                source = branch5.slice();
            }

            // Pick Options At Random From
Current Branch Menu Catalogue
            for(let i = 0; i < 3; i++) {
                let          rand          =
Phaser.Math.Between(0, source.length - 1);
                choices.push(source[rand]);
                source.splice(rand, 1);
            }

            // Card Selection
            stopMenu(pd,             choices,
playOrder[pti], tref);

        } else if (mi[curb][curt] === 1) { //
Landed at Debuff Tile
            console.log("Debuff Tile");

            // Choose Debuff At Random
            let  source  =  ["mun",  "dia",
"kra", "pen", "ket", "peg"];
            let  n  =  Phaser.Math.Between(0,
source.length - 1);
            let choice = source[n];
            stopDebuff(pd,           choice,
playOrder[pti], tref);
```

```
                        } else if (mi[curb][curt] === 2) { //
Landed at Event Tile
                            console.log("Event Tile");

                            // Choose Event At Random
                            let   event   =   ["lmk",   "lsk",
"mia"];
                            let  n  =  Phaser.Math.Between(0,
event.length - 1);
                            let choice = event[n];
                            stopEvent(choice, tref);

                        } else if (mi[curb][curt] === 3) { //
Landed at Money Tile
                            console.log("Money Tile");
                            let            money            =
Phaser.Math.Between(1,3);
                            if (money === 1) {
                                pd[4] += 3;
                            } else if (money === 2) {
                                pd[4] += 4;
                            } else {
                                pd[4] += 5;
                            }

                            console.log("Saving");
                            console.log(pd);
                            if(playOrder[pti] === 1) {
                                tref.game.p1Data           =
pd.slice();
                            } else if (playOrder[pti] === 2)
{
                                tref.game.p2Data           =
pd.slice();
```

```
                                } else if (playOrder[pti] === 3)
{
                                    tref.game.p3Data           =
pd.slice();
                                } else if (playOrder[pti] === 4)
{
                                    tref.game.p4Data           =
pd.slice();
                                }
                                updateText(playOrder[pti],
tref);
                                tref.dice.setScale(2);
                                tref.dice.play('rolling');
                                console.log("ptig :" + pti);
                                turnChange(tref);
                                console.log("ptig :" + pti);
                                updateTurns(playOrder, tref);
                            } else {
                                console.log("Branch Tile");
                                console.log("ptih :" + pti);
                                turnChange(tref);
                                console.log("ptih :" + pti);
                            }
                        }
                    }
                } else if (gameObject === branchB) { // Branch B
                    if (selection === 0) {
                        // Branch 1
                        pd[6] = 1;
                    } else if (selection === 1) {
                        // Branch 3
                        pd[6] = 3;
                    } else if (selection === 2) {
                        // Branch 5
                        pd[6] = 5;
                    }
```

```
console.log("branch aft : " + pd[6]);
console.log(pd);
if (player === 1) {
    tref.game.p1Data = pd.slice();
} else if (player === 2) {
    tref.game.p2Data = pd.slice();
} else if (player === 3) {
    tref.game.p3Data = pd.slice();
} else if (player === 4) {
    tref.game.p4Data = pd.slice();
}
branchA.disableInteractive().setScale(0);
branchB.disableInteractive().setScale(0);
sqText.setScale(0);
sqA1.setScale(0);
sqB1.setScale(0);
if (diceRoll === 0) {
    if (breakpoint) {
        playerMove(playOrder[pti], tref);
        turnChange(tref);
        breakpoint = false;
    }
    tref.dice.setInteractive().setScale(2);
} else {
    if (breakpoint) {
        breakpoint = false;
    }
    let curt = pd[5];
    let curb = pd[6];
    let fin = pd[7];
    while (diceRoll > 0) {
        curt++;
        diceRoll--;
    }
    pd[5] = curt;
    if (player === 1) {
```

```
                    tref.game.p1Data = pd.slice();
                } else if (player === 2) {
                    tref.game.p2Data = pd.slice();
                } else if (player === 3) {
                    tref.game.p3Data = pd.slice();
                } else if (player === 4) {
                    tref.game.p4Data = pd.slice();
                }
                // Stopped Moving / Landed at Tile
                if (diceRoll === 0 && !fin && curt <= 18)
{
                    console.log("Stopped");
                    console.log("pd[5] : " + pd[5]);
                    playerMove(playOrder[pti], tref);

                    if (playerDebuffState[2] < 0) {
                        playerDebuffState[2]--;
                        if (playerDebuffState[2] === 0) {
                            playerDebuffState[0]       =
false;
                        }
                    }
                    if (playerDebuffState[3] < 0) {
                        playerDebuffState[3]--;
                        if (playerDebuffState[3] === 0) {
                            playerDebuffState[1]       =
false;
                        }
                    }
                    // Landed at Menu Tile
                    if (mi[curb][curt] === 0) {
                        console.log("Menu Tile");

                        //  Select  Current  Branch  Menu
Catalogue
                        let choices = [];
```

```
                                let source = [];
                                if (curb === 0) {
                                    source = branch0.slice();
                                } else if (curb === 1) {
                                    source = branch1.slice();
                                } else if (curb === 2) {
                                    source = branch2.slice();
                                } else if (curb === 3) {
                                    source = branch3.slice();
                                } else if (curb === 4) {
                                    source = branch4.slice();
                                } else {
                                    source = branch5.slice();
                                }

                                // Pick Options At Random From
Current Branch Menu Catalogue
                                for(let i = 0; i < 3; i++) {
                                    let          rand          =
Phaser.Math.Between(0, source.length - 1);
                                    choices.push(source[rand]);
                                    source.splice(rand, 1);
                                }

                                // Card Selection
                                stopMenu(pd,            choices,
playOrder[pti], tref);

                            } else if (mi[curb][curt] === 1) { //
Landed at Debuff Tile
                                console.log("Debuff Tile");

                                // Choose Debuff At Random
                                let  source  =  ["mun",  "dia",
"kra", "pen", "ket", "peg"];
```

```
                                let  n  =  Phaser.Math.Between(0,
source.length - 1);
                                let choice = source[n];
                                stopDebuff(pd,           choice,
playOrder[pti], tref);

                            } else if (mi[curb][curt] === 2) { //
Landed at Event Tile
                                console.log("Event Tile");

                                // Choose Event At Random
                                let   event   =   ["lmk",   "lsk",
"mia"];
                                let  n  =  Phaser.Math.Between(0,
event.length - 1);
                                let choice = event[n];
                                stopEvent(choice, tref);

                            } else if (mi[curb][curt] === 3) { //
Landed at Money Tile
                                console.log("Money Tile");
                                let           money           =
Phaser.Math.Between(1,3);
                                if (money === 1) {
                                    pd[4] += 3;
                                } else if (money === 2) {
                                    pd[4] += 4;
                                } else {
                                    pd[4] += 5;
                                }

                                console.log("Saving");
                                console.log(pd);
                                if(playOrder[pti] === 1) {
                                    tref.game.p1Data           =
pd.slice();
```

```
                                } else if (playOrder[pti] === 2)
{
                                    tref.game.p2Data           =
pd.slice();
                                } else if (playOrder[pti] === 3)
{
                                    tref.game.p3Data           =
pd.slice();
                                } else if (playOrder[pti] === 4)
{
                                    tref.game.p4Data           =
pd.slice();
                                }
                                updateText(playOrder[pti],
tref);
                                tref.dice.setScale(2);
                                tref.dice.play('rolling');
                                console.log("ptig :" + pti);
                                turnChange(tref);
                                console.log("ptig :" + pti);
                                updateTurns(playOrder, tref);
                            } else {
                                console.log("Branch Tile");
                                console.log("ptih :" + pti);
                                turnChange(tref);
                                console.log("ptih :" + pti);
                            }
                        }
                    }

                }
            });
        }
```

Fungsi “stopBranch()” adalah fungsi yang bertujuan memproses kondisi pemain untuk menjadi diam jika sebelumnya bergerak dan memilih sebuah cabang untuk dilalui sebelum bisa berlanjut. Untuk memulai fungsi “stopBranch()”, pemain harus berada pada index ke-18 atau

diluar jangkauan informasi petak peta. Ini berarti pemain sedang berada pada sebuah cabang, karena cabang berada diluar *array* petak peta, maka indeks posisi pemain harus berada pada 18 untuk memasuki suatu cabang atau -1 untuk meninggalkan cabang di awal permainan. Proses-proses yang dilakukan fungsi “stopBranch()” meliputi memunculkan dan memasukkan pilihan cabang ke dalam data pemain dan melanjutkan proses evaluasi gerakan pemain jika masih ada gerakan pemain sebelum pemain sampai di petak cabang.

4.2.2 Fungsi Menu

Segmen 4.6 Kode Pilihan Menu

```
// Menu Catalog
        let branch0 = [];
        let branch1 = [];
        let branch2 = [];
        let branch3 = [];
        let branch4 = [];
        let branch5 = [];

       //  Var  Protein, Karbohidrat, Lemak, Kalori, Uang/Biaya,
ID
        let acs = [14.2 , 18.8 , 28.0 , 384, 5, "acs"];    // Ayam
Goreng Cepat Saji
        let ayp = [48.6 , 0.0  , 24.2 , 425, 5, "ayp"];    // Ayam
Panggang
        let bak = [7.0  , 0.0  , 5.0  , 75 , 2, "bak"];     //
Bakso
        let beg = [32.9 , 0.0  , 46.1 , 583, 4, "beg"];     //
Bebek Goreng
        let bis = [2.8  , 17.84, 6.52 , 141, 1, "bis"];     //
Biskuit
        let bcs = [12.9 , 24.5 , 9.36 , 232, 5, "bcs"];     //
Burger Cepat Saji
        let don = [2.12 , 18.8 , 9.96 , 174, 3, "don"];     //
Donat
```

```
        let ecs = [3.82 , 23.8 , 4.37 , 146, 3, "ecs"];    // Es
Krim Cepat Saji
        let hav = [1.07 , 5.27 , 0.25 , 31 , 1, "hav"];    //
Havermut
        let ikl = [6.08 , 0.0  , 2.38 , 48 , 1, "ikl"];    // Ikan
Lele
        let iks = [39.3 , 0.0  , 22.1 , 367, 5, "iks"];    // Ikan
Salmon
        let jas = [4.03 , 20.0 , 1.48 , 108, 2, "jas"];    //
Jagung Segar
        let kah = [5.98 , 15.65, 0.29 , 76 , 1, "kah"];    //
Kacang Hijau
        let kcs = [2.44 , 29.4 , 10.4 , 222, 4, "kcs"];    //
Kentang Goreng Cepat Saji
        let kek = [7    , 38.5 , 21   , 385, 1, "kek"];    //
Keripik Kentang
        let ken = [4.31 , 36.75, 0.19 , 162, 2, "ken"];    //
Kentang
        let keu = [1.86 , 14.02, 9.85 , 151, 1, "keu"];    //
Kerupuk Udang
        let mie = [4.51 , 25.01, 2.06 , 137, 2, "mie"];    // Mie
        let mii = [8.0  , 23.8 , 4.37 , 384, 1, "mii"];    // Mie
Instan
        let nas = [2.69 , 28.2 , 0.28 , 130, 1, "nas"];    // Nasi
        let ncs = [15.0 , 14.3 , 18.8 , 287, 4, "ncs"];    //
Nugget Ayam Cepat Saji
        let piz = [18.3 , 31.4 , 16.6 , 348, 4, "piz"];    //
Pizza
        let pop = [12.9 , 77.8 , 4.54 , 387, 5, "pop"];    //
Popcorn
        let rop = [7.49 , 30.73, 2.15 , 167, 1, "rop"];    // Roti
Putih
        let sak = [6.05 , 11.62, 3.24 , 100, 1, "sak"];    // Sari
Kedelai
        let sin = [1.63 , 45.72, 0.34 , 192, 1, "sin"];    //
Singkong
```

```
        let sos = [7.83 , 0.15 , 12    , 143, 1, "sos"];     //
Sosis Sapi
        let sts = [26.2 , 0     , 4.28 , 150, 3, "sts"];     //
Steak Sapi
        let sus = [6.56 , 9.34 , 6.4  , 120, 2, "sus"];    // Susu
Sapi
        let tag = [18.8 , 8.86 , 20.2 , 270, 1, "tag"];    // Tahu
Goreng
        let teg = [6.26 , 0.38 , 6.81 , 90 , 1, "teg"];     //
Telur Goreng
        let teo = [6.47 , 0.39 , 7.14 , 94 , 1, "teo"];     //
Telur Omelet
        let tmg = [19.9 , 7.62 , 11.4 , 195, 1, "tmg"];     //
Tempe Goreng
        let uds = [7.04 , 0.0   , 0.18 , 30 , 1, "uds"];     //
Udang Segar

        branch0.push(nas, hav, rop, teg, sak, sus);
        branch1.push(acs, bcs, ecs, kcs, ncs, pop);
        branch2.push(beg, jas, mie, nas, ikl, iks);
        branch3.push(bis, don, kek, keu, tag, tmg);
        branch4.push(ayp, kah, ken, sin, teo, uds);
        branch5.push(bak, mii, piz, sos, sts);
```

Segmen 4.6 berisi pilihan menu yang mungkin dipilih secara acak oleh sistem dan ditunjukkan ke pemain dalam satu cabang. Setiap cabang memiliki pilihan menu yang berbeda untuk mencerminkan pilihan makanan yang mungkin ditemukan dalam situasi yang sesuai dengan konteks cabang tersebut.

Segmen 4.7 Kode Fungsi Menu

```
// Menu Tile Process
        function stopMenu(pd, choices, player, tref) {
            console.log("stopMenu");
            console.log("pd " + pd);
            let mon = pd[4];
            let playerDebuffState;
            if (player === 1) {
```

```
                playerDebuffState = tref.game.p1Debuff[0];
            } else if (player === 2) {
                playerDebuffState = tref.game.p2Debuff[0];
            } else if (player === 3) {
                playerDebuffState = tref.game.p3Debuff[0];
            } else if (player === 4) {
                playerDebuffState = tref.game.p4Debuff[0];
            }

            // Display Options
            let select = cardSelect(1, choices, tref);
            let skipBtn = tref.add.rectangle(512, 600, 200, 100,
0xffffff).setInteractive().setDepth(10);
            let skipTxt = tref.add.text(skipBtn.x, skipBtn.y,
"Skip", {
                fontFamily: 'Arial Black', fontSize: 38, color:
'#ffffff',
                stroke: '#000000', strokeThickness: 8,
                align: 'center'
            }).setOrigin(0.5).setDepth(10);

            // Player Input
            let chosen;
            tref.input.on('gameobjectdown', function(pointer,
gameObject) {
                // Card Selection and Money Check
                if ((gameObject === select[0] && mon <
choices[0][4]) ||
                (gameObject === select[1] && mon < choices[1][4])
||
                (gameObject === select[2] && mon <
choices[2][4])) {
                    // Play sound buzzer for not enough money
                } else if (gameObject === select[0] && mon >=
choices[0][4]) {
                    // Menu Option 1
```

```
                    chosen = choices[0];
                    pd = addMenu(pd, chosen, playerDebuffState);
                    console.log("Menu 1");
                    select.forEach(element => {

element.disableInteractive().setScale(0);
                    });
                    skipBtn.disableInteractive().setScale(0);
                    skipTxt.setScale(0);
                    console.log("pd " + pd);
                    console.log('playerorder : ' + player);

                    // Save Changes
                    if(player === 1) {
                        tref.game.p1Data = pd.slice();
                    } else if (player === 2) {
                        tref.game.p2Data = pd.slice();
                    } else if (player === 3) {
                        tref.game.p3Data = pd.slice();
                    } else if (player === 4) {
                        tref.game.p4Data = pd.slice();
                    }
                    updateText(player, tref);
                    tref.dice.setScale(2);
                    tref.dice.play('rolling');

                    // Change Turns
                    // console.log("ptia :" + pti);
                    turnChange(tref);
                    // console.log("ptia :" + pti);
                    updateTurns(tref.playOrder2, tref);

                } else if (gameObject === select[1] && mon >=
choices[1][4]) {
                    // Menu Option 2
                    chosen = choices[1];
```

```
                    pd = addMenu(pd, chosen, playerDebuffState);
                    console.log("Menu 2");
                    select.forEach(element => {

element.disableInteractive().setScale(0);
                    });
                    skipBtn.disableInteractive().setScale(0);
                    skipTxt.setScale(0);
                    console.log("pd " + pd);
                    console.log('playerorder : ' + player);

                    // Save Changes
                    if(player === 1) {
                        tref.game.p1Data = pd.slice();
                    } else if (player === 2) {
                        tref.game.p2Data = pd.slice();
                    } else if (player === 3) {
                        tref.game.p3Data = pd.slice();
                    } else if (player === 4) {
                        tref.game.p4Data = pd.slice();
                    }
                    updateText(player, tref);
                    tref.dice.setScale(2);
                    tref.dice.play('rolling');

                    // Change Turns
                    // console.log("ptib :" + pti);
                    turnChange(tref);
                    // console.log("ptib :" + pti);
                    updateTurns(tref.playOrder2, tref);

                } else if (gameObject === select[2] && mon >=
choices[2][4]) {
                    // Menu Option 3
                    chosen = choices[2];
                    pd = addMenu(pd, chosen, playerDebuffState);
```

```
                    console.log("Menu 3");
                    select.forEach(element => {

element.disableInteractive().setScale(0);
                    });
                    skipBtn.disableInteractive().setScale(0);
                    skipTxt.setScale(0);
                    console.log("pd " + pd);
                    console.log('playerorder : ' + player);

                    // Save Changes
                    if(player === 1) {
                        tref.game.p1Data = pd.slice();
                    } else if (player === 2) {
                        tref.game.p2Data = pd.slice();
                    } else if (player === 3) {
                        tref.game.p3Data = pd.slice();
                    } else if (player === 4) {
                        tref.game.p4Data = pd.slice();
                    }
                    updateText(player, tref);
                    tref.dice.setScale(2);
                    tref.dice.play('rolling');

                    // Change Turns
                    // console.log("ptic :" + pti);
                    turnChange(tref);
                    // console.log("ptic :" + pti);
                    updateTurns(tref.playOrder2, tref);

                } else if (gameObject === skipBtn) {
                    // Skip Button
                    console.log("Skip Button");
                    select.forEach(element => {

element.disableInteractive().setScale(0);
```

```
            });
            skipBtn.disableInteractive().setScale(0);
            skipTxt.setScale(0);
            tref.dice.setScale(2);
            tref.dice.play('rolling');

            // Change Turns
            // console.log("ptid :" + pti);
            turnChange(tref);
            // console.log("ptid :" + pti);
            updateTurns(tref.playOrder2, tref);

        }
    });
  }
// Menu Value Add Processing
  function addMenu(pd, chosen, dia) {
    if (!dia) { // Not debuffed (diarrhea)
        pd[0] += chosen[0];
        pd[1] += chosen[1];
        pd[2] += chosen[2];
        pd[3] += chosen[3];
    } else { // Debuffed (diarrhea)
        pd[0] += Math.round((chosen[0] / 2) * 100) / 100;
        pd[1] += Math.round((chosen[1] / 2) * 100) / 100;
        pd[2] += Math.round((chosen[2] / 2) * 100) / 100;
        pd[3] += Math.round(chosen[3] / 2);
    }
    pd[0] = Math.round(pd[0] * 100) / 100;
    pd[1] = Math.round(pd[1] * 100) / 100;
    pd[2] = Math.round(pd[2] * 100) / 100;
    pd[4] -= chosen[4];
    return pd;
  }
function getRatio(tref) {
    let data, ratio;
```

```
        if (playOrder[pti] === 1) {
            data = tref.game.p1Data;
            ratio = tref.game.p1Ratio;
        } else if (playOrder[pti] === 2) {
            data = tref.game.p2Data;
            ratio = tref.game.p2Ratio;
        } else if (playOrder[pti] === 3) {
            data = tref.game.p3Data;
            ratio = tref.game.p3Ratio;
        } else if (playOrder[pti] === 4) {
            data = tref.game.p4Data;
            ratio = tref.game.p4Ratio;
        }
        console.log(data);
        let total = data[0] + data[1] + data[2];
        if (total != 0) {
            ratio[0] = Math.round(data[0] * 100 / total);
            ratio[1] = Math.round(data[1] * 100 / total);
            ratio[2] = Math.round(data[2] * 100 / total);
        } else {
            ratio[0] = 0;
            ratio[1] = 0
            ratio[2] = 0;
        }
        console.log(ratio);
    }
```

Fungsi “stopMenu()” adalah fungsi yang bertujuan menampilkan pilihan menu yang tersedia di cabang tersebut kepada pemain. Tiga kartu menu yang dipilih secara acak dari kemungkinan pilihan menu di cabang tersebut ditunjukkan kepada pemain untuk dipilih. Saat pemain memilih menu makanan, maka status yang terdapat pada kartu tersebut akan ditambahkan ke pemain. Tetapi, jika pemain sedang mengidap sebuah *debuff* yang berupa diare, maka hanya setengah gizi yang dapat diambil pemain. Terakhir, rasio keseimbangan gizi pemain akan dicek kembali dan tampilan status pemain akan diperbarui.

4.2.3 Fungsi Debuff

Segmen 4.8 Kode Fungsi Debuff

```
let bd0 = ['pen', 'kra'];
let bd1 = ['mun', 'dia'];
let bd2 = ['ket', 'kra'];
let bd3 = ['peg', 'dia'];
let bd4 = ['mun', 'dia'];
let bd5 = ['ket', 'kra'];

// Debuff Tile Process
function stopDebuff(pd, player, tref) {
    console.log("stopDebuff");
    let choices;
    if (pd[6] === 0) {
        choices = bd0;
    } else if (pd[6] === 1) {
        choices = bd1;
    } else if (pd[6] === 2) {
        choices = bd2;
    } else if (pd[6] === 3) {
        choices = bd3;
    } else if (pd[6] === 4) {
        choices = bd4;
    } else if (pd[6] === 5) {
        choices = bd5;
    }
    let rand = Phaser.Math.Between(0,1);
    let debuff = choices[rand];
    let select = cardSelect(2, debuff, tref);
    let playerDebuff;
    let playerRef;

    // Get Player References
    if(player === 1) {
        playerDebuff = tref.game.p1Debuff;
```

```
                playerRef = tref.p1Info;
            } else if (player === 2) {
                playerDebuff = tref.game.p2Debuff;
                playerRef = tref.p2Info;
            } else if (player === 3) {
                playerDebuff = tref.game.p3Debuff;
                playerRef = tref.p3Info;
            } else if (player === 4) {
                playerDebuff = tref.game.p4Debuff;
                playerRef = tref.p4Info;
            }
            console.log(playerDebuff);

            // Player Input
            let mode = 1;
            tref.input.on('gameobjectdown',    function(pointer,
gameObject) {
                if (gameObject === select && mode === 1) {
                    console.log("debuff : " + debuff);
                    if (debuff === "mun") { // Muntah
                        if (pd[0] === 0 || pd[1] === 0 || pd[2]
=== 0) {
                            pd[8] -= 200;
                        }
                        pd[0] -= 20;
                        pd[1] -= 40;
                        pd[2] -= 10;
                    } else if (debuff === "dia") { // Diare
                        playerDebuff[0] = true;
                        playerDebuff[2] = 3;
                        pd[8] -= 100;
                        playerRef[8].setColor('#ff0000');
                    } else if (debuff === "kra") { // Kram
                        playerDebuff[1] = true;
                        playerDebuff[3] = 3;
                        pd[8] -= 100;
```

```
                    playerRef[9].setColor('#ff0000');
                } else if (debuff === "pen") { // Pendarahan
                    pd[0] -= 50;
                    pd[8] -= 100;
                } else if (debuff === "ket") { // Ketosis
                    pd[2] -= 25;
                    pd[8] -= 100;
                } else if (debuff === "peg") { // Penyakit
Graves
                    pd[1] -= 100;
                    pd[8] -= 100;
                }
                pd[0] = Math.round(pd[0] * 100) / 100;
                pd[1] = Math.round(pd[1] * 100) / 100;
                pd[2] = Math.round(pd[2] * 100) / 100;

                // If Below 0 Set to 0
                if (pd[0] < 0) {
                    pd[0] = 0;
                }
                if (pd[1] < 0) {
                    pd[1] = 0;
                }
                if (pd[2] < 0) {
                    pd[2] = 0;
                }

                // Save Changes
                if(player === 1) {
                    tref.game.p1Data = pd.slice();
                    tref.game.p1Debuff                    =
playerDebuff.slice()
                } else if (player === 2) {
                    tref.game.p2Data = pd.slice();
                    tref.game.p2Debuff                    =
playerDebuff.slice()
```

```
                    } else if (player === 3) {
                        tref.game.p3Data = pd.slice();
                        tref.game.p3Debuff                    =
playerDebuff.slice()
                    } else if (player === 4) {
                        tref.game.p4Data = pd.slice();
                        tref.game.p4Debuff                    =
playerDebuff.slice()
                    }

                    console.log(playerDebuff);
                    select.disableInteractive().setScale(0);
                    updateText(player, tref);
                    leadCheck(tref);
                    tref.dice.setScale(2);
                    tref.diceText.setScale(1);

                    // Change Turns
                    turnChange(tref);
                    updateTurns(tref.playOrder2, tref);
                }
            });
        }
```

Fungsi *debuff* memberikan *debuff* kepada pemain, dimana efek *debuff* sesuai dengan kartu yang dipilih secara acak oleh sistem. Pilihan kartu *debuff* tergantung dari cabang pemain, dengan setiap cabang memiliki kemungkinan *debuff* yang berbeda. Kartu *debuff* kemudian memiliki dua jenis, yaitu kartu yang memiliki giliran dan kartu yang berefek langsung. Kartu *debuff* yang memiliki giliran disimpan menjadi variabel *debuff* status pemain dan dihitung berapa lama *debuff* itu masih berpengaruh.

4.2.4 Fungsi Event

Segmen 4.9 Kode Fungsi Event

```
// Event Tile Process
        function stopEvent(event, tref) {
```

```
            console.log("stopEvent");
            let select = cardSelect(3, event, tref);
            let mode = 2;

            // Event Start
            tref.input.on('gameobjectdown',    function(pointer,
gameObject) {
                if (gameObject === select && mode === 2) {
                    select.setScale(0).disableInteractive();
                    console.log("Event Started");
                    tref.turnsText.setScale(0);
                    tref.pturnText.setScale(0);
                    let pTurn = 1;
                    let                  pText                  =
tref.add.text(tref.game.canvas.width/2, 10, "Player " + pTurn +
"\'s Turn");
                    pText.setFontFamily('Arial
Black').setFontSize(38).setColor('#ffffff').setStroke('#000000',
8).setAlign('center');
                    pText.x -= pText.width/2;
                    let                eventDice                =
tref.add.sprite(tref.game.canvas.width/2,
tref.game.canvas.height/2, 'dice').setScale(2);
                    eventDice.setInteractive();
                    eventDice.play("rolling");
                    let maxRoll = 0;
                    let maxPlayer = [];

                    // Player Input
                    tref.input.on('gameobjectdown',
function(pointer, gameObject) {
                        // Player 1
                        if (gameObject === eventDice && pTurn ===
1) {
                            let diceRand = Phaser.Math.Between(1,
6);
```

```
                            eventDice.play("slowing");
                            eventDice.disableInteractive();
                            tref.time.delayedCall(1500, () => {
                                eventDice.stop();
                                eventDice.setFrame(diceRand    -
1);
                                if (maxRoll < diceRand) {
                                    maxRoll = diceRand;
                                    maxPlayer.length = 0;
                                    maxPlayer.push(pTurn);
                                } else if (maxRoll === diceRand)
{
                                    maxPlayer.push(pTurn);
                                }
                                if (event === "lmk") {
                                    tref.game.p1Data[0]       +=
diceRand;
                                    tref.game.p1Data[1]       +=
diceRand * 13;
                                    tref.game.p1Data[3]       +=
diceRand * 55;
                                } else if (event === "lsk") {
                                    tref.game.p1Data[3]       -=
diceRand * 100;
                                } else if (event === "mia") {
                                    if (diceRand > 3) {
                                        tref.game.p1Debuff     =
[false,false, 0, 0];
                                    }
                                }
                                tref.time.delayedCall(1000,    ()
=> {
                                    updateText(pTurn, tref);
                                    pTurn++;
                                    pText.setText("Player   "   +
pTurn + "\'s Turn");
```

```
                        eventDice.setInteractive();
                        eventDice.play("rolling");
                    });
                });
            }

            // Player 2
            if (gameObject === eventDice && pTurn ===
2) {
                let diceRand = Phaser.Math.Between(1,
6);
                eventDice.play("slowing");
                eventDice.disableInteractive();
                tref.time.delayedCall(1500, () => {
                    eventDice.stop();
                    eventDice.setFrame(diceRand    -
1);
                    if (maxRoll < diceRand) {
                        maxRoll = diceRand;
                        maxPlayer.length = 0;
                        maxPlayer.push(pTurn);
                    } else if (maxRoll === diceRand)
{
                        maxPlayer.push(pTurn);
                    }
                    if (event === "lmk") {
                        tref.game.p2Data[0]       +=
diceRand;
                        tref.game.p2Data[1]       +=
diceRand * 13;
                        tref.game.p2Data[3]       +=
diceRand * 55;
                    } else if (event === "lsk") {
                        tref.game.p2Data[3]       -=
diceRand * 100;
                    } else if (event === "mia") {
```

```
                                    if (diceRand > 3) {
                                        tref.game.p2Debuff     =
[false,false, 0, 0];
                                    }
                                }
                                tref.time.delayedCall(1500,    ()
=> {
                                    updateText(pTurn, tref);
                                    pTurn++;
                                    pText.setText("Player   "   +
pTurn + "\'s Turn");
                                    eventDice.setInteractive();
                                    eventDice.play("rolling");
                                });
                            });
                        }

                        // Player 3
                        if (gameObject === eventDice && pTurn ===
3) {
                            let diceRand = Phaser.Math.Between(1,
6);
                            eventDice.play("slowing");
                            eventDice.disableInteractive();
                            tref.time.delayedCall(1500, () => {
                                eventDice.stop();
                                eventDice.setFrame(diceRand    -
1);
                                if (maxRoll < diceRand) {
                                    maxRoll = diceRand;
                                    maxPlayer.length = 0;
                                    maxPlayer.push(pTurn);
                                } else if (maxRoll === diceRand)
{
                                    maxPlayer.push(pTurn);
                                }
```

```
                                if (event === "lmk") {
                                    tref.game.p3Data[0]       +=
diceRand;
                                    tref.game.p3Data[1]       +=
diceRand * 13;
                                    tref.game.p3Data[3]       +=
diceRand * 55;
                                } else if (event === "lsk") {
                                    tref.game.p3Data[3]       -=
diceRand * 100;
                                } else if (event === "mia") {
                                    if (diceRand > 3) {
                                        tref.game.p3Debuff     =
[false,false, 0, 0];
                                    }
                                }
                                tref.time.delayedCall(1000,    ()
=> {
                                    updateText(pTurn, tref);
                                    pTurn++;
                                    pText.setText("Player   "   +
pTurn + "\'s Turn");
                                    eventDice.setInteractive();
                                    eventDice.play("rolling");
                                });
                            });
                        }

                        // Player 4
                        if (gameObject === eventDice && pTurn ===
4) {
                            let diceRand = Phaser.Math.Between(1,
6);
                            eventDice.play("slowing");
                            eventDice.disableInteractive();
                            tref.time.delayedCall(1500, () => {
```

```
                                eventDice.stop();
                                eventDice.setFrame(diceRand     -
1);
                                if (maxRoll < diceRand) {
                                    maxRoll = diceRand;
                                    maxPlayer.length = 0;
                                    maxPlayer.push(pTurn);
                                } else if (maxRoll === diceRand)
{
                                    maxPlayer.push(pTurn);
                                }
                                if (event === "lmk") {
                                    tref.game.p4Data[0]        +=
diceRand;
                                    tref.game.p4Data[1]        +=
diceRand * 13;
                                    tref.game.p4Data[3]        +=
diceRand * 55;
                                } else if (event === "lsk") {
                                    tref.game.p4Data[3]        -=
diceRand * 100;
                                } else if (event === "mia") {
                                    if (diceRand > 3) {
                                        tref.game.p4Debuff      =
[false,false, 0, 0];
                                    }
                                }
                                tref.time.delayedCall(1000,    ()
=> {
                                    pText.setScale(0);

eventDice.setScale(0).disableInteractive();
                                    eventGivePoint(event,
maxPlayer, tref);
                                    tref.turnsText.setScale(1);
                                    tref.pturnText.setScale(1);
```

```
                                    updateText(pTurn, tref);
                                    tref.dice.setScale(2);
                                    tref.dice.play('rolling');
                                    //  console.log("ptif  :"  +
pti);
                                    turnChange(tref);
                                    //  console.log("ptif  :"  +
pti);
                                    updateTurns(tref.playOrder2,
tref);

                                });
                            });
                        }
                    });
                }
            });
        }

        // Event Scores Processing
        function eventGivePoint(event, maxPlayer, tref) {
            console.log("eventGivePoint");
            // console.log("event: " + event);
            // console.log("Player " + maxPlayer);
            let arr = [];
            for (let i = 0; i < maxPlayer.length; i++) {
                arr.push(maxPlayer[i]);
            }
            if (event === "lmk" || event === "lsk") {
                for (let i = 0; i < arr.length; i++) {
                    if (arr[i] === 1) {
                        tref.game.p1Data[8] += 100;
                        updateText(1, tref);
                    } else if (arr[i] === 2) {
                        tref.game.p2Data[8] += 100;
                        updateText(2, tref);
```

```
                } else if (arr[i] === 3) {
                    tref.game.p3Data[8] += 100;
                    updateText(3, tref);
                } else if (arr[i] === 4) {
                    tref.game.p4Data[8] += 100;
                    updateText(4, tref);
                }
            }
        } else if (event === "mia") {
            tref.game.p1Data[8] += 50;
            tref.game.p2Data[8] += 50;
            tref.game.p3Data[8] += 50;
            tref.game.p4Data[8] += 50;
            updateText(1, tref);
            updateText(2, tref);
            updateText(3, tref);
            updateText(4, tref);
        }
    }
```

Fungsi *event* memulai sebuah *event* sesuai dengan kartu yang terpilih. Ketika sebuah *event* dimulai, dadu yang sebelumnya digunakan untuk pergerakan pemain diganti menjadi dadu *event* yang bergilir secara manual ke setiap pemain. Lama eksekusi *event* tergantung dari jumlah pemain dalam permainan. Di akhir *event* pemain dengan lemparan dadu tertinggi diberi bonus skor tergantung *event*-nya.

4.3 Pergerakan Pemain

4.3.1 Implementasi

Game beroperasi pada sistem input pengguna yang berarti game dimulai dan diakhiri oleh input pengguna. Pada awal game, pemain diminta untuk memilih sebuah cabang dan melempar dadu, dan pemain akan melempar dadu untuk maju dan akhirnya sampai di petak akhir peta. Berdasarkan hal tersebut, pergerakan pemain dimulai dengan input dadu pemain.

Segmen 4.10 Core Loop Dice Input

```
// First Player Branch Selection
```

```
        stopBranch(playOrder[pti], 0, this);
        let breakpoint = false;

        // Dice Onclick Trigger
        this.input.on('gameobjectdown',    function    (pointer,
gameObject) {
            if (gameObject === tref.dice) {
                diceRoll = Phaser.Math.Between(1,6);
                tref.diceText.setScale(0);
                // diceRoll = 6;
                tref.dice.play('rolling');
                tref.dice.disableInteractive();
                console.log("diceroll " + diceRoll);
                // Debuff Check : Kram
                // Jika diceRoll lebih dari satu maka kurangi satu
                if (diceRoll > 1) {
                    if (playOrder[pti] === 1) {
                        if (tref.game.p1Debuff[1]) {
                            diceRoll--;
                        }
                    } else if (playOrder[pti] === 2) {
                        if (tref.game.p2Debuff[1]) {
                            diceRoll--;
                        }
                    } else if (playOrder[pti] === 3) {
                        if (tref.game.p3Debuff[1]) {
                            diceRoll--;
                        }
                    } else if (playOrder[pti] === 4) {
                        if (tref.game.p4Debuff[1]) {
                            diceRoll--;
                        }
                    }
                }
                console.log("diceroll " + diceRoll);
                // Mengurangi Turn Debuff
```

```
if (playOrder[pti] === 1) {
    if (tref.game.p1Debuff[0]) {
        tref.game.p1Debuff[2]--;
        if (tref.game.p1Debuff[2] === 0) {
            tref.p1Info[8].setColor('#d9d9d9');
            tref.game.p1Debuff[0] = false;
        } else {
            tref.p1Info[8].setColor('#ff0000');
        }
    }
    if (tref.game.p1Debuff[1]) {
        tref.game.p1Debuff[3]--;
        if (tref.game.p1Debuff[3] === 0) {
            tref.game.p1Debuff[1] = false;
            tref.p1Info[9].setColor('#d9d9d9');
        } else {
            tref.p1Info[9].setColor('#ff0000');
        }
    }
} else if (playOrder[pti] === 2) {
    if (tref.game.p2Debuff[0]) {
        tref.game.p2Debuff[2]--;
        if (tref.game.p2Debuff[2] === 0) {
            tref.game.p2Debuff[0] = false;
            tref.p2Info[8].setColor('#d9d9d9');
        } else {
            tref.p2Info[8].setColor('#ff0000');
        }
    }
    if (tref.game.p2Debuff[1]) {
        tref.game.p2Debuff[3]--;
        if (tref.game.p2Debuff[3] === 0) {
            tref.game.p2Debuff[1] = false;
            tref.p2Info[9].setColor('#d9d9d9');
        } else {
            tref.p2Info[9].setColor('#ff0000');
```

```
            }
        }
    } else if (playOrder[pti] === 3) {
        if (tref.game.p3Debuff[0]) {
            tref.game.p3Debuff[2]--;
            if (tref.game.p3Debuff[2] === 0) {
                tref.game.p3Debuff[0] = false;
                tref.p3Info[8].setColor('#d9d9d9');
            } else {
                tref.p3Info[8].setColor('#ff0000');
            }
        }
        if (tref.game.p3Debuff[1]) {
            tref.game.p3Debuff[3]--;
            if (tref.game.p3Debuff[3] === 0) {
                tref.game.p3Debuff[1] = false;
                tref.p3Info[9].setColor('#d9d9d9');
            } else {
                tref.p3Info[9].setColor('#ff0000');
            }
        }
    } else if (playOrder[pti] === 4) {
        if (tref.game.p4Debuff[0]) {
            tref.game.p4Debuff[2]--;
            if (tref.game.p4Debuff[2] === 0) {
                tref.game.p4Debuff[0] = false;
                tref.p4Info[8].setColor('#d9d9d9');
            } else {
                tref.p4Info[8].setColor('#ff0000');
            }
        }
        if (tref.game.p4Debuff[1]) {
            tref.game.p4Debuff[3]--;
            if (tref.game.p4Debuff[3] === 0) {
                tref.game.p4Debuff[1] = false;
                tref.p4Info[9].setColor('#d9d9d9');
```

```
                        } else {
                            tref.p4Info[9].setColor('#ff0000');
                        }
                    }
                }

                tref.time.delayedCall(3000, () => {
                    tref.dice.stop();
                    // Play Sound
                    tref.dice.setFrame(diceRoll - 1);

                    tref.time.delayedCall(1500, () => {
                        tref.dice.setScale(0);
                        tref.dice.setInteractive();

                        let playerData, playerDebuffState;
                        if (playOrder[pti] === 1) {
                            playerData = tref.game.p1Data;
                            playerDebuffState                   =
tref.game.p1Debuff;
                        } else if (playOrder[pti] === 2) {
                            playerData = tref.game.p2Data;
                            playerDebuffState                   =
tref.game.p2Debuff;
                        } else if (playOrder[pti] === 3) {
                            playerData = tref.game.p3Data;
                            playerDebuffState                   =
tref.game.p3Debuff;
                        } else if (playOrder[pti] === 4) {
                            playerData = tref.game.p4Data;
                            playerDebuffState                   =
tref.game.p4Debuff;
                        }

                        //  playerMove(playOrder[pti],  diceRoll,
tref);
```

```
                        while (diceRoll > 0) {
                            // Move Player
                            playerData[5] += 1;
                            diceRoll--;

                            if (mi[playerData[6]][playerData[5]]
=== 3) {
                                console.log("Money          Tile
Passed");
                                playerData[4] += 1;
                                if  (playerData[6]  ===  1  ||
playerData[6] === 2) {
                                    playerData[4] += 2;
                                }
                            }
                            console.log("Saving");
                            console.log(playerData);
                            if(playOrder[pti] === 1) {
                                tref.game.p1Data               =
playerData.slice();
                                tref.game.p1Debuff             =
playerDebuffState;
                                updateText(1, tref);
                            } else if (playOrder[pti] === 2) {
                                tref.game.p2Data               =
playerData.slice();
                                tref.game.p2Debuff             =
playerDebuffState;
                                updateText(2, tref);
                            } else if (playOrder[pti] === 3) {
                                tref.game.p3Data               =
playerData.slice();
                                tref.game.p3Debuff             =
playerDebuffState;
                                updateText(3, tref);
                            } else if (playOrder[pti] === 4) {
```

```
                                tref.game.p4Data                =
playerData.slice();
                                tref.game.p4Debuff              =
playerDebuffState;
                                updateText(4, tref);
                            }

                            // Finish Check
                            if    (playerData[5]    >=    18    &&
((playerData[6] === 4) || playerData[6] === 5)) {
                                playerData[7] = true;
                                change = true;
                                breakpoint = true;
                                if(playOrder[pti] === 1) {
                                    tref.game.p1Data            =
playerData.slice();
                                } else if (playOrder[pti] === 2)
{
                                    tref.game.p2Data            =
playerData.slice();
                                } else if (playOrder[pti] === 3)
{
                                    tref.game.p3Data            =
playerData.slice();
                                } else if (playOrder[pti] === 4)
{
                                    tref.game.p4Data            =
playerData.slice();
                                }
                                turnChange(tref);
                                break;
                            } else if (playerData[5] >= 18) {
                                // Branch Check
                                breakpoint = true;
                                stopBranch(playOrder[pti],
diceRoll, tref);
```

```
                        break;
                    }
                }
                console.log("stopCheck");
                console.log("diceroll : " + diceRoll);
                console.log("finish : " + playerData[7]);
                console.log("breakpoint     :     "     +
breakpoint);
                // Stopped Moving / Landed at Tile
                if  (diceRoll  ===  0  &&  !playerData[7]
&& !breakpoint) {
                    console.log("Stopped");
                    console.log("pd[5]     :     "     +
playerData[5]);
                    playerMove(playOrder[pti], tref);
                    if (playerDebuffState[2] < 0) {
                        playerDebuffState[2]--;
                        if (playerDebuffState[2] === 0) {
                            playerDebuffState[0]       =
false;
                        }
                    }
                    if (playerDebuffState[3] < 0) {
                        playerDebuffState[3]--;
                        if (playerDebuffState[3] === 0) {
                            playerDebuffState[1]       =
false;
                        }
                    }
                    // Landed at Menu Tile
                    if  (mi[playerData[6]][playerData[5]]
=== 0) {
                        console.log("Menu Tile");

                        //  Select  Current  Branch  Menu
Catalogue
```

```
                                let choices = [];
                                let source = [];
                                if (playerData[6] === 0) {
                                    source = branch0.slice();
                                } else if (playerData[6] === 1) {
                                    source = branch1.slice();
                                } else if (playerData[6] === 2) {
                                    source = branch2.slice();
                                } else if (playerData[6] === 3) {
                                    source = branch3.slice();
                                } else if (playerData[6] === 4) {
                                    source = branch4.slice();
                                } else {
                                    source = branch5.slice();
                                }

                                // Pick Options At Random From
Current Branch Menu Catalogue
                                for(let i = 0; i < 3; i++) {
                                    let          rand          =
Phaser.Math.Between(0, source.length - 1);
                                    choices.push(source[rand]);
                                    source.splice(rand, 1);
                                }

                                // Card Selection
                                stopMenu(playerData,    choices,
playOrder[pti], tref);

                            }              else              if
(mi[playerData[6]][playerData[5]] === 1) { // Landed at Debuff
Tile
                                console.log("Debuff Tile");
                                stopDebuff(playerData,
playOrder[pti], tref);
```

```
                            }               else                if
(mi[playerData[6]][playerData[5]] === 2) { // Landed at Event Tile
                                console.log("Event Tile");

                                // Choose Event At Random
                                let event;
                                if (playerData[6] % 2 === 0) {
                                    event   =   ['lsk',   'lmk',
'mia'];
                                } else if (playerData[6] % 2 ===
1) {
                                    event = ['lmk', 'mia'];
                                } else {
                                    event = ['mia'];
                                }
                                let  n  =  Phaser.Math.Between(0,
event.length - 1);
                                let choice = event[n];
                                stopEvent(choice, tref);

                            }               else                if
(mi[playerData[6]][playerData[5]] === 3) { // Landed at Money Tile
                                console.log("Money Tile");
                                let           money           =
Phaser.Math.Between(1,3);
                                if (money === 1) {
                                    playerData[4] += 3;
                                } else if (money === 2) {
                                    playerData[4] += 4;
                                } else {
                                    playerData[4] += 5;
                                }
                                if   (playerData[6]   ===   1   ||
playerData[6] === 2) {
                                    playerData[4] += 2;
                                }
```

```
                                console.log("Saving");
                                console.log(playerData);
                                if(playOrder[pti] === 1) {
                                    tref.game.p1Data           =
playerData.slice();
                                } else if (playOrder[pti] === 2)
{
                                    tref.game.p2Data           =
playerData.slice();
                                } else if (playOrder[pti] === 3)
{
                                    tref.game.p3Data           =
playerData.slice();
                                } else if (playOrder[pti] === 4)
{
                                    tref.game.p4Data           =
playerData.slice();
                                }
                                updateText(playOrder[pti],
tref);
                                tref.dice.setScale(2);
                                // tref.dice.play('rolling');
                                tref.diceText.setScale(1);
                                console.log("ptig :" + pti);
                                turnChange(tref);
                                console.log("ptig :" + pti);
                                updateTurns(playOrder, tref);
                            } else {
                                console.log("Branch Tile");
                                console.log("ptih :" + pti);
                                turnChange(tref);
                                console.log("ptih :" + pti);
                            }
                        }
                    });
```

```
            });
        }
    });
```

Gameplay loop dimulai dengan input pemain melempar dadu. Sebelum pemain berjalan, dilakukan pengecekan untuk kemungkinan *debuff* bergilir yang dimiliki pemain untuk memperbarui status mereka, baik dalam mengurangi sisa giliran efek *debuff* atau mematikannya. Lemparan dadu kemudian melalui *loop* dimana hasil lemparan dadu akan dikurangi satu persatu hingga mencapai nol. Di dalam *loop* tersebut juga terdapat pengecekan untuk petak uang dimana jika pemain hanya melewati petak uang pemain masih diberikan uang walaupun lebih sedikit daripada jika pemain berhenti di petak uang. Setelah lemparan dadu menjadi nol, sistem akan memanggil fungsi yang berhubungan dengan aksi yang akan dilakukan di petak pemain saat itu, seperti memanggil kartu menu di petak menu dan memilih cabang di petak cabang. Di dalam fungsi-fungsi aksi ini kemudian fungsi di bagian 4.1.4 yang mengubah giliran pemain dan melanjutkan permainan hingga tidak ada lagi pemain yang tersisa di dalam *array* giliran pemain, mengakhiri permainan.

4.3.2 Akhir Game

Segmen 4.11 Kode Akhir Game

```
        // Scoring Process

        let data = [this.game.p1Data, this.game.p2Data,
this.game.p3Data, this.game.p4Data];
        let ratios = [this.game.p1Ratio, this.game.p2Ratio,
this.game.p3Ratio, this.game.p4Ratio];
        let scores = [this.game.p1Data[8], this.game.p2Data[8],
this.game.p3Data[8], this.game.p4Data[8]];

       // Bonus Score Checking
        for (let i = 0; i <= this.game.players - 1; i++) {
            console.log("data");
            console.log(data[i]);

            console.log(data[i][3]);
            if (data[i][3] >= 2000) {
```

```
        console.log('caltrue');
        scores[i] += 500;
        briefs[i][3].setColor('#00ff00');
        briefs[i][7].setText('Score : 500');
    } else {
        console.log('calfalse');
        briefs[i][3].setColor('#ff0000');
        briefs[i][7].setText('Score : 0');
    }

    if (10 <= ratios[i][0] && ratios[i][0] <= 35) {
        console.log('protrue');
        scores[i] += 300;
        briefs[i][0].setColor('#00ff00');
        briefs[i][4].setText('Score : 300');
    } else {
        console.log('profalse');
        briefs[i][0].setColor('#ff0000');
        briefs[i][4].setText('Score : 0');
    }
    if (45 <= ratios[i][1] && ratios[i][1] <= 65) {
        console.log('cartrue');
        scores[i] += 300;
        briefs[i][1].setColor('#00ff00');
        briefs[i][5].setText('Score : 300');
    } else {
        console.log('carfalse');
        briefs[i][1].setColor('#ff0000');
        briefs[i][5].setText('Score : 0');
    }
    if (20 <= ratios[i][2] && ratios[i][2] <= 35) {
        console.log('fattrue');
        scores[i] += 300;
        briefs[i][2].setColor('#00ff00');
        briefs[i][6].setText('Score : 300');
    } else {
```

```
                console.log('fatfalse');
                briefs[i][2].setColor('#ff0000');
                briefs[i][6].setText('Score : 0');
            }
            if (data[i][4] < 0) {
                console.log('debttrue');
                scores[i] += data[i][4] * 50;
            }
        }

        score1.setText('Player 1 Score : ' + scores[0]);
        if (data[0][4] < 0) {
            score1.setText('Player 1 Score : ' + scores[0] + "
(Debt : " + (data[0][4] * 50) + ")");
        }

        let max = scores[0];
        let winner = 1;

        if (this.game.players > 1) {
            score2.setText('Player 2 Score : ' + scores[1]);
            if (data[1][4] < 0) {
                score2.setText('Player 2 Score : ' + scores[1] +
" (Debt : " + (data[1][4] * 50) + ")");
            }
            if (max < scores[1]) {
                max = scores[1];
                winner = 2;
            }
        }
        if (this.game.players > 2) {
            score3.setText('Player 3 Score : ' + scores[2]);
            if (data[2][4] < 0) {
                score3.setText('Player 3 Score : ' + scores[2] +
" (Debt : " + (data[2][4] * 50) + ")");
            }
```

```
            if (max < scores[2]) {
                max = scores[2];
                winner = 3;
            }
        }
        if (this.game.players > 3) {
            score4.setText('Player 4 Score : ' + scores[3]);
            if (data[3][4] < 0) {
                score4.setText('Player 4 Score : ' + scores[3] +
" (Debt : " + (data[3][4] * 50) + ")");
            }
            if (max < scores[3]) {
                max = scores[3];
                winner = 4;
            }
        }

        this.add.text(512, this.game.canvas.height - 50, 'Player
' + winner + ' Wins!', {
            fontFamily: 'Arial Black', fontSize: 64, color:
'#ffff00',
            stroke: '#000000', strokeThickness: 8,
            align: 'center'
        }).setOrigin(0.5);
```

Skor pemain ditentukan oleh jumlah skor mereka dalam game dan kemudian dari persentase keseimbangan gizi mereka dan apakah mereka memenuhi kebutuhan rata-rata energi dalam sehari. Untuk keseimbangan gizi, 300 poin ditambahkan ketika salah satu jenis gizi terpenuhi keseimbangannya dengan maksimum 900 bonus poin dari keseimbangan gizi. Kemudian akan ditambahkan lagi 500 poin jika energi atau kalori yang didapatkan pemain lebih dari kebutuhan rata-rata manusia dalam sehari yaitu 2000 kcal. Terakhir, poin akan dikurangi sebanyak 50 dikali dengan jumlah uang yang dihutang oleh pemain. Pemain dengan skor atau poin terbesar memenangkan permainan.