DOCUMENTATION

Code Besonderheiten & Lösungen

Levelgenerator:

Um jede Runde ein neues Level generieren zu können, baue ich mir ein zwei dimensionales Array (32x32 pixel per arrayindex) über das Spielfeld, das meine statischen Objekte und Leerfelder beinhaltet. Und lasse dieses jede Runde neu erstellen. Dabei kann ich gut verschiedene Wahrscheinlichkeiten, Abstände von Objekten, Positionen und maximale Anzahl festlegen. (zb. Den minimalen Abstand von Kakteen zu bestimmen, durch einen neighbourcheck)

```
/* 0 = nothing
 * 1 = start
 * 2 = end
 * 3 = enemy
 * 4 = quicksand
```

```
// enemys
if (row > 1 && row < 25 && column > 7 && column < 51 && enemyCounter < 10)
{
    neighbourCheck = false;

    for (int i = row - 2; i < row + 3; i++) // checks neighboured tiles
    {
        for (int j = column - 3; j < column + 4; j++)
        {
            if (level[i, j] == 3)
            {
                neighbourCheck = true;
            }
        }
    }

    if (!neighbourCheck)
    {
        if (GetChance(989, enemyChance))
        {
            level[row, column] = 3;
            enemyCounter++;
        }
    }
}
```

Für mehr Randomness habe ich vor allem eine kleine Methode GetChance geschrieben, die mir zurückgibt ob eine gewisse Chance eingetroffen ist.

```
// returns true if the chance hit, false if not
public static bool GetChance(int max, int chance)
{
    int aRandom = rnd.Next(0, max);
    if (aRandom < chance)
    {
        return true;
    }
    return false;
}
```

Kontrolle & Respawn der Kakteengeschosse

Um meherer Objekte gleichzeitig in verschiedene Richtungen zu bewegen lasse ich wieder Coordinaten Arrays erzeugen, die rund um den Kakteen 4 Projectiles spawnen lassen. Da ich weis in welcher Reihenfolge sie erzeugt werden, kann ich diese leicht in verschiedene richtungen updaten, bzw. wieder respawnen falls sie das Spielfeld verlassen und somit gut in for loops handlen. Bei jedem Update checke ich auch gleich ob eine Collision mit den Spielern besteht.

```
//********* updating cactusghosts & collision & respawn *********
if (currentState == GameState.gameScreen)
{
    int projectileArrayCounter = 0;

    for (int i = 0; i < cactusProjectileArray.GetLength(0); i++)
    {
        if (projectileArrayCounter == 0) // left movement
        {
            cactusProjectileArray[i, 0] -= playerSpeed * deltaTime;
            if (cactusProjectileArray[i,0] < 176)
            {
                cactusProjectileArray[i, 0] = cactusProjectileStartArray[i, 0];
            }
        }
        if (projectileArrayCounter == 1) // up movement
        {
            cactusProjectileArray[i, 1] -= playerSpeed * deltaTime;
            if (cactusProjectileArray[i, 1] < 108)
            {
                cactusProjectileArray[i, 1] = cactusProjectileStartArray[i, 1];
            }
        }
```

Rundenwiederholung

Um eine neue Runde einzuleiten wird einfach eine Methode aufgerufen, die alles beinhaltet was ich neu erzeugen und zurücksetzen möchte. Natürlich mit Random generierten Startpositionen der Spieler und einem neuen random Spielfeld. Diese kommt mir dann auch beim kompletten Resetten des Spiels zu gute.

```
// ******** NEW ROUND **********
if (currentState == GameState.gameScreen
    && (isDeadPlayerOne == true || inGoalPlayerOne == true)
    && (isDeadPlayerTwo == true || inGoalPlayerTwo == true)
    && (isDeadPlayerThree == true || inGoalPlayerThree == true)
    && (isDeadPlayerFour == true || inGoalPlayerFour == true)
    && endTimer <= 0
    && roundsCounter > 0)
{
    newRound();
    roundsCounter--;
    currentRound++;
}

void newRound()
{
    // returns players to start position
    startingPositions = new int[4];
    HelperMethods.FillArrayRandomSingleOccurrence(startingPositions);
    HelperMethods.StartPositionToX(startingPositions);
    playerOnePosition = new Vector2(88, startingPositions[0]);
    playerTwoPosition = new Vector2(88, startingPositions[1]);
    playerThreePosition = new Vector2(88, startingPositions[2]);
    playerFourPosition = new Vector2(88, startingPositions[3]);
    // gets players back to life
    isDeadPlayerOne = false;
    isDeadPlayerTwo = false;
    isDeadPlayerThree = false;
    isDeadPlayerFour = false;
```

Menuführung

Für die Menuführung habe ich mir eine simple Lösung aus eigenen Menustates gebastelt. Je nach derzeitigem Menustate, kann die Rundenanzahl verändert werden, oder das Spiel gestartet. Je nach Menustate ändert sich auch die Anzeige in dem Draw. Da die Joisticks der Controller jeweils die Bereiche von -1/+1 in x und y Richtung abdecken muss immer daran gedacht werden die abgenommenen float werte bei bedarf nach int zu casten.

```
private void MenuPlayer(float deltaTime, GamePadCapabilities plC, GamePadState state)
{
    if (plC.HasLeftXThumbStick)
    {
        menuNav -= state.ThumbSticks.Left.Y * 0.1f;
        if (menuState == 1)
        {
            roundsWannaPlay += state.ThumbSticks.Left.X * 0.1f;
        }

    }
}
```

```
// ********* menü ************
if (roundsWannaPlay <= 1)
{
    roundsWannaPlay = 1;
}
if (roundsWannaPlay > 11)
{
    roundsWannaPlay = 10;
}
roundsToPlay = (int)roundsWannaPlay;
```

```
if (menuNav >= 4)
{
    menuNav = 0;
}
if (menuNav < 0)
{
    menuNav = 3.9f;
}
menuState = (int)menuNav;
```

```
if (menuState == 0)
{
    spriteBatch.DrawString(menutext, "Play", new Vector2(880, 200), Color.FromNonPremultiplied(209, 107, 239, 255));
    spriteBatch.DrawString(menutext, "Rounds " + roundsToPlay, new Vector2(800, 350), Color.FromNonPremultiplied(254, 255, 238, 255));
    spriteBatch.DrawString(menutext, "Options", new Vector2(825, 500), Color.FromNonPremultiplied(254, 255, 238, 255));
    spriteBatch.DrawString(menutext, "Quit", new Vector2(880, 650), Color.FromNonPremultiplied(254, 255, 238, 255));
}
else if (menuState == 1)
{
    spriteBatch.DrawString(menutext, "Play", new Vector2(880, 200), Color.FromNonPremultiplied(254, 255, 238, 255));
    spriteBatch.DrawString(menutext, "Rounds " + roundsToPlay, new Vector2(800, 350), Color.FromNonPremultiplied(209, 107, 239, 255));
    spriteBatch.DrawString(menutext, "Options", new Vector2(825, 500), Color.FromNonPremultiplied(254, 255, 238, 255));
    spriteBatch.DrawString(menutext, "Quit", new Vector2(880, 650), Color.FromNonPremultiplied(254, 255, 238, 255));
}
```

Timer & Countdowns

Timer und Countdowns die nach der eingestellten deltaTime geupdated werden, sind super Werkzeuge um einen Spielablauf zu gestalten, oder Menupausen zu erzwingen. Damit jedes Menu mit dem gleichen Knopf bedient werden kann, fand ich die Lösung eines kurzen Timers am Besten, der verhindert, direkt vom Intro Screen in den Game Screen zu rutschen.

```
// **** timer press a ******
if (hasPressed)
{
    hasPressedTimer -= 1.3f * deltaTime;
    if (hasPressedTimer < 0)
    {
        hasPressed = false;
    }
}
if (hasPressed == false && gamePadOneState.Buttons.A == ButtonState.Pressed)
{
    hasPressedTimer = 1.0f;
    hasPressed = true;
}
```

```
        if (gamePadOneState.Buttons.A == ButtonState.Pressed && menuState == 0 && hasPressedTimer < 0)
        {
            currentState = GameState.learnScreen;
        }
        if (gamePadOneState.Buttons.A == ButtonState.Pressed && menuState == 3 && hasPressedTimer < 0)
        {
            Exit();
        }
    }
    break;
case GameState.learnScreen:
    {
        if (gamePadOneState.Buttons.A == ButtonState.Pressed && hasPressedTimer < 0)
        {
            currentState = GameState.gameScreen;
```