Game name: Squish Sprint

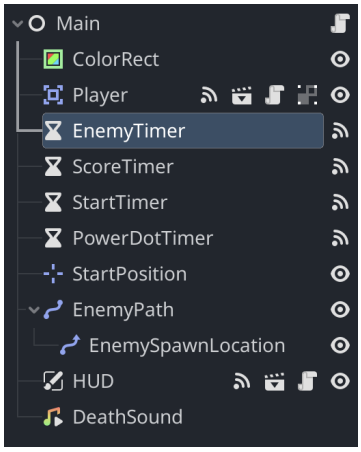
Implemented functionality:
Player blob moves around the screen while avoiding randomly spawned blue blobs. The green player blob moves around in any direction using the WASD as up, left, down, and right (can be combined to travel diagonally). When the green blob player gets hit by a blue blob enemy, the player dies. When the green blob player hovers over the rat power up, it gets 10 extra points each time. However, when the blue blobs touch the rat, they can push the rat off the screen and can no longer be accessed for power up points. The game ends when the player gets touched by a blue blob and the score is counted where the player gets points for every second he stays alive plus the additonal bonus points from the rat power up. This score is displayed at the top of the game and updated as the score increases during gameplay. To start the game, there is a main menu with a start button to start the actual game that the player clicks and goes away when the game starts. All the sprites (player blob, enemy blob, and rat power up) are animated based on their actions and whether they are idle or moving. The player blob has sound effects for different events (squishy sounds for when it is moving and a classic death sound for when it gets killed by the blue blobs). The blue blob enemies are randomly spawned NPC's that are set on a random trajectory and travel through the screen in a straight line in hopes of killing the green blob.

How to play:
Click start game when ready to play. Move the character up, left, down, and right with the WASD buttons respectively to dodge the blue blobs. Hover over the rat before the blue blobs knock him off the screen to get 10 extra points each time and get 1 point for every second you stay alive (score show at top).

Link to video footage: https://youtu.be/lhZFQK9-mZs

Code layout:
At a high level, the game is made of a main scene that is composed of various nodes and prefabs.

Main Scene
First, there is a main scene that is made of a generic node that has children as a combination of a ColorRect node to set the background as a light pink, a Player prefab (Area2D node) to spawn in and set the traits/interactions of the user controlled green blob, different timers for the enemy spawning, counting the score, starting the game, and spawning the powerup rat, a Marker2D node to set the starting spawning point of the green blob, a Path2D and PathFollow2D node to spawn the enemies randomly around the screen, a CanvasLayer node to set up the HUD, and an AudioStreamPlayer to create the death sound effects when the blob dies and the game ends. A screen shot of that scene hierarchy is pasted below:



As for the code in the script for the main scene, the main pieces of code are the game loop control and sending various signals. We have a NewGame() method that removes the sprites from the previous round and begins to setup for the next game. This setup includes resetting the score to 0, spawning the green player blob at the starting position, and displaying the main menu HUD.

```
private void NewGame()
{
    // removes old enemies and dots from previous attempt
    GetTree().CallGroup("enemies", Node.MethodName.QueueFree);
    GetTree().CallGroup("powerdots", Node.MethodName.QueueFree);

    _score = 0;

    var player = GetNode<Player>("Player");
    var startPosition = GetNode<Marker2D>("StartPosition");
    player.Start(startPosition.Position);

    GetNode<Timer>("StartTimer").Start();

    var hud = GetNode<HUD>("HUD");
    hud.UpdateScore(_score);
    hud.ShowMessage("Get Ready!");
}
```

While the game is starting up, a signal is sent to start the start timer. This timer starts many all the other timers which will perform various actions as they timeout.

```
private void OnStartTimerTimeout()
{
    GetNode<Timer>("EnemyTimer").Start();
    GetNode<Timer>("ScoreTimer").Start();
    GetNode<Timer>("PowerDotTimer").Start();
}
```

On the timeout of the enemy timer it spawns in 1 new enemy blob. It performs different actions to randomize the spawning location, the spawning direction, and the spawning velocity.

```
private void OnEnemyTimerTimeout()
{
    // create new instance of Enemy scene
    Enemy enemy = EnemyScene.Instantiate<Enemy>();

    // choose random location on Path2D
    var enemySpawnLocation = GetNode<PathFollow2D>("EnemyPath/EnemySpawnLocation");
    enemySpawnLocation.ProgressRatio = GD.Randf();

    // set enemy's direction perpendicular to path direction
    float direction = enemySpawnLocation.Rotation + Mathf.Pi / 2;

    // set enemy's position to random location
    enemy.Position = enemySpawnLocation.Position;

    // add randomness to direction
    direction += (float) GD.RandRange(-Mathf.Pi / 4, Mathf.Pi / 4);
    enemy.Rotation = direction;

    // choose velocity
    var velocity = new Vector2((float) GD.RandRange(150.0, 250.0), 0);
    enemy.LinearVelocity = velocity.Rotated(direction);

    // spawn enemy by adding it to main scene
    AddChild(enemy);
}
```

On the timeout of the score timer, it simply increments the score by 1 to indicate the player survived for another second and gets another point. It also updates this incrementation by displaying the updated score in the HUD.

```
private void OnScoreTimerTimeout()
{
    _score++;
    GetNode<HUD>("HUD").UpdateScore(_score);
}
```

On the timeout of the powerdot timer, it does a one shot spawning of the powerup rat at a randomly chosen location of 100, 100.

```
private void OnPowerDotTimerTimeout()
{
    // create new instance of Dot scene
    PowerDot dot = PowerDotScene.Instantiate<PowerDot>();

    // set dot's position to random location
    dot.Position = new Vector2(100, 100);

    // spawn dot by adding it to main scene
    AddChild(dot);
}
```

We also have a GameOver() method that takes care of the game when the blob dies. Upon getting killed, a signal is sent that runs this method, which in turn stops the enemy timer, the score timer, and the powerdot timer. The game over HUD is displayed briefly while the game over sound is played.

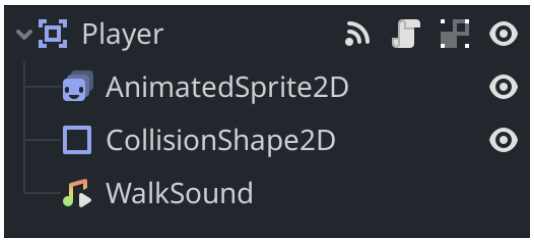
```
private void GameOver()
{
    GetNode<Timer>("EnemyTimer").Stop();
    GetNode<Timer>("ScoreTimer").Stop();
    GetNode<Timer>("PowerDotTimer").Stop();

    GetNode<HUD>("HUD").ShowGameOver();

    GetNode<AudioStreamPlayer>("DeathSound").Play();
}
```

Player Scene

Next, there is a Player scene that is an Area2D node and is used as a prefab in the main scene. Its children include an AnimatedSprinte2D node that sets the blob image as its animation, a CollisionShape2D node that sets the collision boundaries for the blob character, and a AudioStreamPlayer node that allows the blob to make squishy noises when he moves around. A screen shot of that scene hierarchy is pasted below:



The bulk of the player script code is just depicting all the interactions of the player blob in the Process method. This includes reading in input from the WASD keys to move the player, taking in those same clicks to play noises, playing animations based on movement speed, and tracking the location of the player to keep it within the screen bounds.

```
// called every frame
public override void _Process(double delta)
{
    var velocity = Vector2.Zero; // movement speed

    // move player with WASD
    if (Input.IsActionPressed("move_up"))
    {
        velocity.Y -= 1;
    }
    if (Input.IsActionPressed("move_down"))
    {
        velocity.Y += 1;
    }
    if (Input.IsActionPressed("move_left"))
    {
        velocity.X -= 1;
    }
    if (Input.IsActionPressed("move_right"))
    {
        velocity.X += 1;
    }
```

```
if (Input.IsActionJustPressed("move_up"))
{
    GetNode<AudioStreamPlayer>("WalkSound").Play();
}
if (Input.IsActionJustReleased("move_up"))
{
    GetNode<AudioStreamPlayer>("WalkSound").Stop();
}
if (Input.IsActionJustPressed("move_down"))
{
    GetNode<AudioStreamPlayer>("WalkSound").Play();
}
if (Input.IsActionJustReleased("move_down"))
{
    GetNode<AudioStreamPlayer>("WalkSound").Stop();
}
if (Input.IsActionJustPressed("move_left"))
{
    GetNode<AudioStreamPlayer>("WalkSound").Play();
}
if (Input.IsActionJustReleased("move_left"))
{
    GetNode<AudioStreamPlayer>("WalkSound").Stop();
}
if (Input.IsActionJustPressed("move_right"))
{
    GetNode<AudioStreamPlayer>("WalkSound").Play();
}
if (Input.IsActionJustReleased("move_right"))
{
    GetNode<AudioStreamPlayer>("WalkSound").Stop();
}
```

```
// animate sprite based on movement
var animatedSprite2D = GetNode<AnimatedSprite2D>("AnimatedSprite2D");

if (velocity.Length() > 0)
{
    velocity = velocity.Normalized() * Speed;
    animatedSprite2D.Play();
}
else
{
    animatedSprite2D.Stop();
}
```

```
// contain player within screen
Position += velocity * (float) delta;
Position = new Vector2(
    x: Mathf.Clamp(Position.X, 0, ScreenSize.X),
    y: Mathf.Clamp(Position.Y, 0, ScreenSize.Y)
);
```

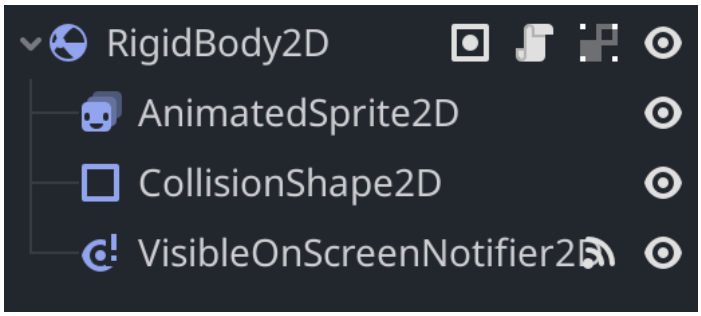
There is also a OnBodyEntered function that does the stuff with collisions. It detects whether the collision is from the power dot where it emits a signal to add scores and update that on the HUD or whether the collision is from the enemy blobs where it emits a signal to kill the blob and end the game.

```
private void OnBodyEntered(PhysicsBody2D body)
{

    if ("PowerDot" == body.Name)
    {
        EmitSignal(SignalName.PowerHit);
    }
    else
    {
        Hide();
        EmitSignal(SignalName.Hit);
        GetNode<CollisionShape2D>("CollisionShape2D").SetDeferred(CollisionShape2D.PropertyName.Disabled, true);
    }
}
```

Enemy Scene

Next, there is a Enemy scene that is a RigidBody2D node and is used as a prefab in the main scene although not in the scene hierarchy directly because it's added in runtime. Its children include an AnimatedSprinte2D node that sets the blob image as its animation, a CollisionShape2D node that sets the collision boundaries for the blob enemies, and a VisibleOnScreenNotifier2D node that allows the game to delete enemies that have exited the screen bounds. A screen shot of that scene hierarchy is pasted below:

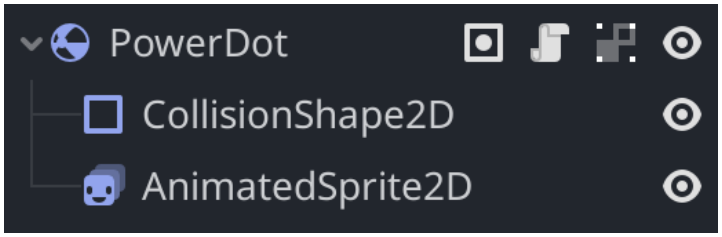


The code is simply a Ready method that starts the animation of the enemies and a signal handler that deletes the enemy blob if it has exited the screen.

```
public partial class Enemy : RigidBody2D
{
    // Called when the node enters the scene tree for the first time.
    public override void _Ready()
    {
        var animatedSprite2D = GetNode<AnimatedSprite2D>("AnimatedSprite2D");
        animatedSprite2D.Play();
    }

    private void OnVisibleOnScreenNotifier2DScreenExited()
    {
        QueueFree();
    }
}
```

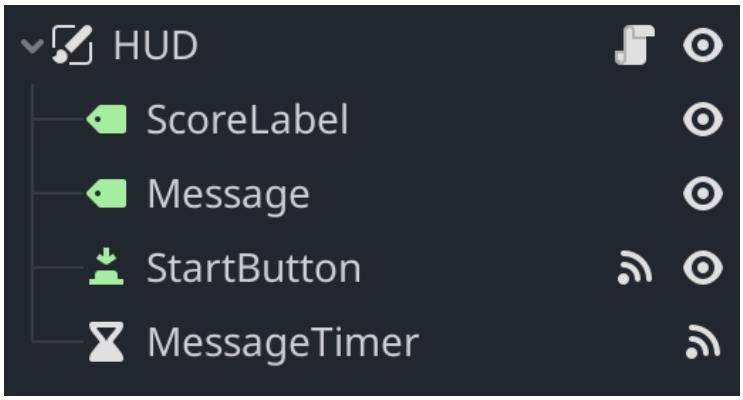
PowerDot Scene
Next, there is a PowerDot scene that is a RigidBody2D node and is used as a prefab in the main scene, also being added in runtime rather than beforehand. Its children include an AnimatedSprinte2D node that sets the rat image and its animation and a CollisionShape2D node that sets the collision boundaries for the rat powerup character. A screen shot of that scene hierarchy is pasted below:



The code for the powerdot is just the Ready method that starts the animation for the rat image.

```
public partial class PowerDot : RigidBody2D
{
    // Called when the node enters the scene tree for the first time.
    public override void _Ready()
    {
        var animatedSprite2D = GetNode<AnimatedSprite2D>("AnimatedSprite2D");
        animatedSprite2D.Play();
    }
}
```

HUD Scene
Lastly, there is a HUD scene that is a CanvasLayer node and is used as a prefab in the main scene. Its children include a Label node for the score, a Label node for displaying various messages on the screen, a Button node to add a start button to the main menu, and a timer that spaces out the timing of displaying messages. A screen shot of that scene hierarchy is pasted below:



The code includes a ShowMessage method and a ShowGameOver method that are used at different parts of the game to show messages on the screen. The ShowMessage method is simply used to display whatever message is passed in until the timer runs out. The ShowGameOver method is used to display the Game Over message as well as set up the main menu with the Squish Sprint message and the start button.

```
public void ShowMessage(string text)
{
    var message = GetNode<Label>("Message");
    message.Text = text;
    message.Show();

    GetNode<Timer>("MessageTimer").Start();
}

async public void ShowGameOver()
{
    ShowMessage("Game Over");

    var messageTimer = GetNode<Timer>("MessageTimer");
    await ToSignal(messageTimer, Timer.SignalName.Timeout);

    var message = GetNode<Label>("Message");
    message.Text = "Squish Sprint";
    message.Show();

    await ToSignal(GetTree().CreateTimer(1.0), SceneTreeTimer.SignalName.Timeout);
    GetNode<Button>("StartButton").Show();
}
```

There is also a UpdateScore method that gets called when the score has been changed and needs to be updated on the actual screen.

```
public void UpdateScore(int score)
{
    GetNode<Label>("ScoreLabel").Text = score.ToString();
}
```

The OnStartButtonPressed method sends out the signal that starts up the actual game and runs the method that sets up a new game.

```
private void OnStartButtonPressed()
{
    GetNode<Button>("StartButton").Hide();
    EmitSignal(SignalName.StartGame);
}
```