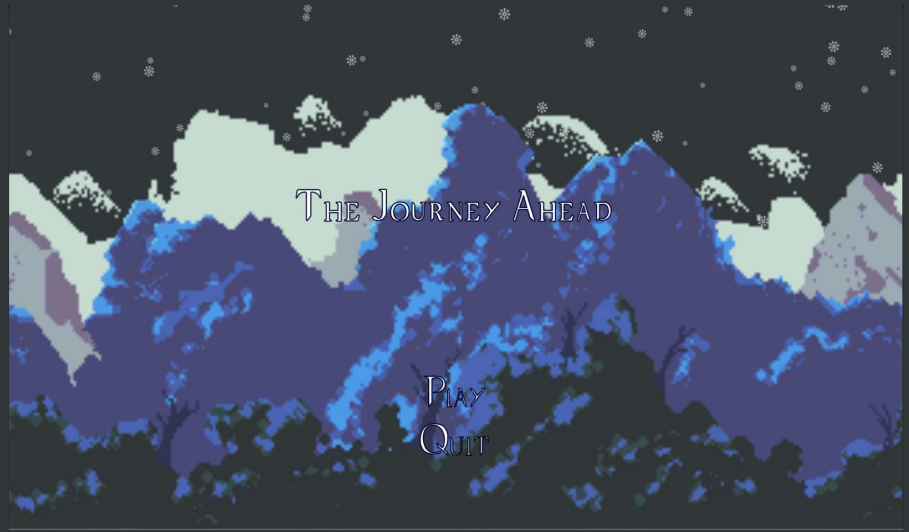

The Journey Ahead
By: William Valentin

What is The Journey Ahead?

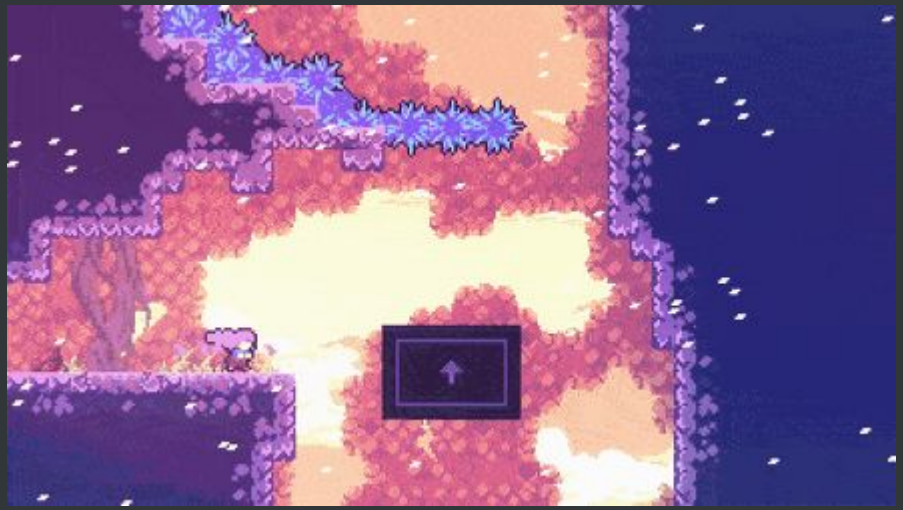
The Journey ahead is a pixel style 2D platformer that focuses on movement mechanics and maneuvering the player character across different platforms while evading and attacking enemies along the way with fluid gameplay.



Inspirations

Hollow Knight

Celeste



Roles

- Game Design
- Level Design
- Programmer
- Animator
- Audio sfx

Goals

- Work efficiently in a 2D environment in Unity.
- Animate character and enemies for different states.
- Snappy movement and controls
- Death/Damaging
- Consistent level design
- Fluid gameplay

Movement and Mechanics

Movement Controls

- A and D keys to move left and right
- Space key to jump
- Mouse click to attack
- Direction press along wall to slide/wall jump

```
// Input from the player
1 reference
private void CheckInput()
{
    movementInputDirection = Input.GetAxisRaw("Horizontal");

    isRunning = movementInputDirection != 0;

    anim.SetBool("isRunning", isRunning);

    if(Input.GetButtonDown("Jump"))
    {
        if (isGrounded || (amountOfJumpsLeft > 0 && isTouchingWall))
        {
            NormalJump();
            jumpSoundEffect.Play();
        }
        else
        {
            jumpTimer = jumpTimerSet;
            isAttemptingToJump = true;
        }
    }

    if(Input.GetButtonDown("Horizontal") && isTouchingWall)
    {
        if(!isGrounded && movementInputDirection != facingDirection)
        {
            canMove = false;
            canFlip = false;

            turnTimer = turnTimerSet;
        }
    }

    if (!canMove)
    {
        turnTimer -= Time.deltaTime;

        if(turnTimer <= 0)
        {
            canMove = true;
            canFlip = true;
        }
```

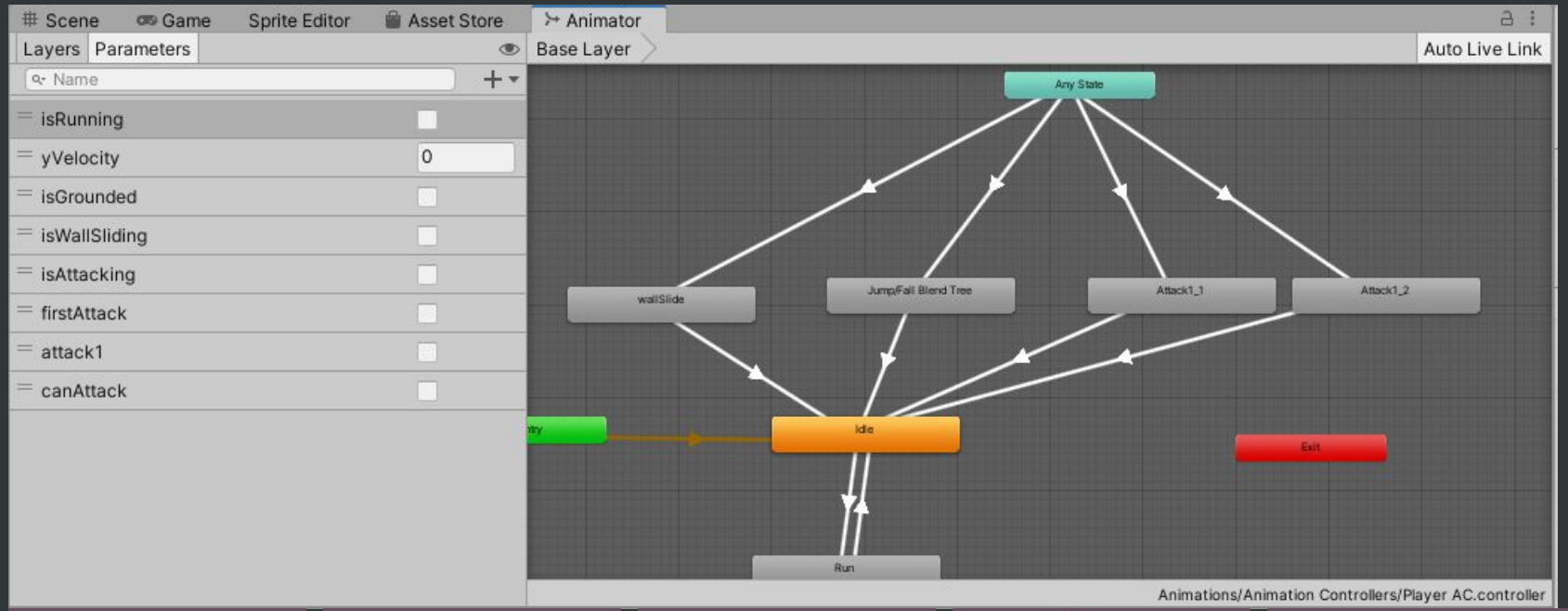
```
1 reference
private void WallJump()
{
    if (canWallJump)
    {
        rb.velocity = new Vector2(rb.velocity.x, 0.0f);
        isWallSliding = false;
        amountOfJumpsLeft = amountOfJumps;
        amountOfJumpsLeft--;
        Vector2 forceToAdd = new Vector2(wallJumpForce * wallJumpDirection.x * movementInputDirection, wallJumpForce * wallJumpDirection.y);
        rb.AddForce(forceToAdd, ForceMode2D.Impulse);
        jumpTimer = 0;
        isAttemptingToJump = false;
        checkJumpMultiplier = true;
        turnTimer = 0;
        canMove = true;
        canFlip = true;
        hasWallJumped = true;
        wallJumpTimer = wallJumpTimerSet;
        lastWallJumpDirection = -facingDirection;
    }
}

// Applying movement to player
1 reference
private void ApplyMovement()
{

    if (!isGrounded && !isWallSliding && movementInputDirection == 0 && !knockback)
    {
        rb.velocity = new Vector2(rb.velocity.x * airDragMultiplier, rb.velocity.y);
    }


    else if(canMove && !knockback)
    {
        rb.velocity = new Vector2(movementSpeed * movementInputDirection, rb.velocity.y);
    }

    //rb.velocity = new Vector2(movementSpeed * movementInputDirection, rb.velocity.y);

    if (isWallSliding)
    {
        if(rb.velocity.y < -wallSlideSpeed)
        {
            rb.velocity = new Vector2(rb.velocity.x, -wallSlideSpeed);
        }
    }
}
```

Enemy States

- Detecting the player at specified distance
- Following player
- Attacking the player
- Damaging player
- Patrolling areas
- Taking damage

```
2 references
public override void Start()
{
    base.Start();

    moveState = new E1_MoveState(this, stateMachine, "move", moveStateData, this);
    idleState = new E1_IdleState(this, stateMachine, "idle", idleStateData, this);
    playerDetectedState = new E1_PlayerDetectedState(this, stateMachine, "playerDetected", playerDetectedData, this);
    chargeState = new E1_ChargeState(this, stateMachine, "charge", chargeStateData, this);
    lookForPlayerState = new E1_LookForPlayerState(this, stateMachine, "lookForPlayer", lookForPlayerStateData, this);
    meleeAttackState = new E1_MeleeAttackState(this, stateMachine, "meleeAttack", meleeAttackPosition, meleeAttackStateData, this);
    stunState = new E1_StunState(this, stateMachine, "stun", stunStateData, this);
    deadState = new E1_DeadState(this, stateMachine, "dead", deadStateData, this);

    stateMachine.Initialize(moveState);
}

2 references
public override void OnDrawGizmos()
{
    base.OnDrawGizmos();

    Gizmos.DrawWireSphere(meleeAttackPosition.position, meleeAttackStateData.attackRadius);
}

2 references
public override void Damage(AttackDetails attackDetails)
{
    base.Damage(attackDetails);

    if (isDead)
    {
        stateMachine.ChangeState(deadState);
    }
    else if (isStunned && stateMachine.currentState != stunState)
    {
        stateMachine.ChangeState(stunState);
    }
}
}
```

Animations



Idle

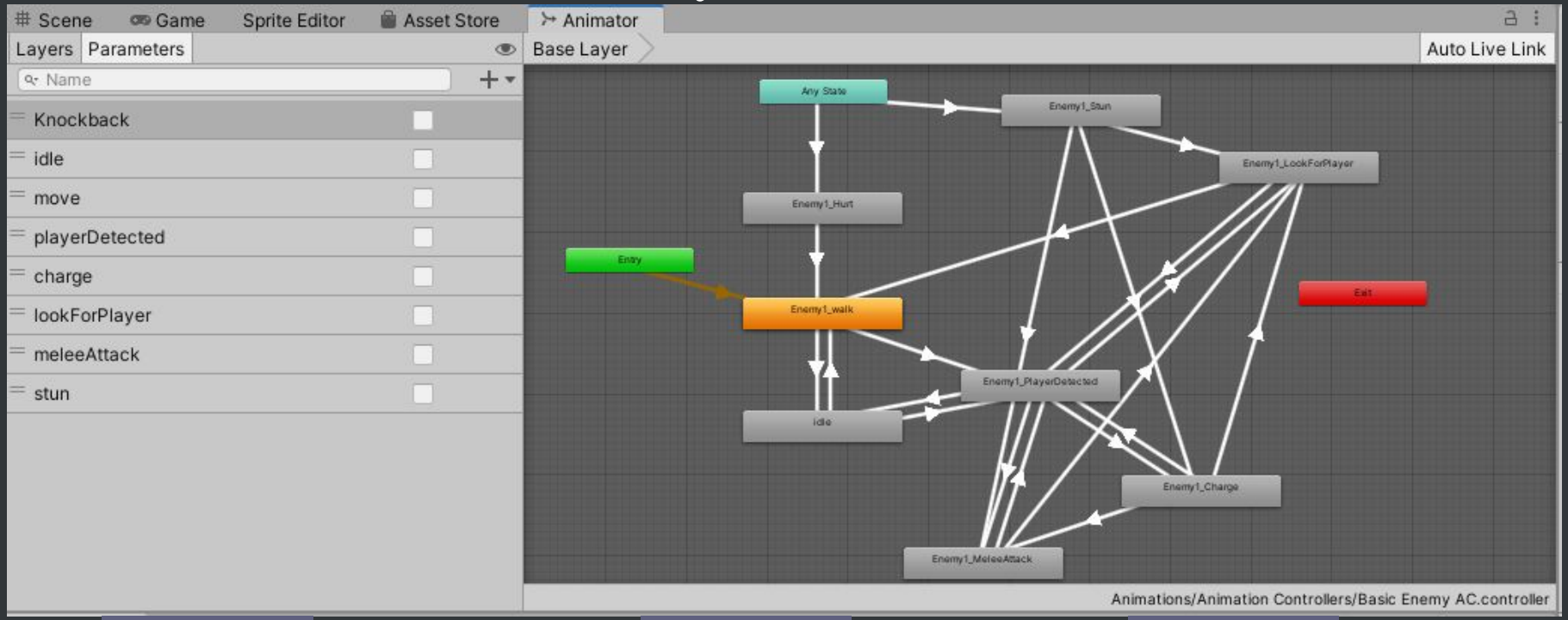
Running

Jump/fall

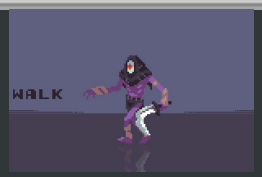
Attacking

WallSlide

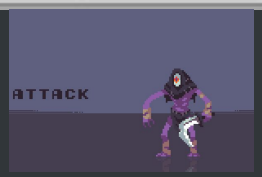
Enemy Animations



Idle

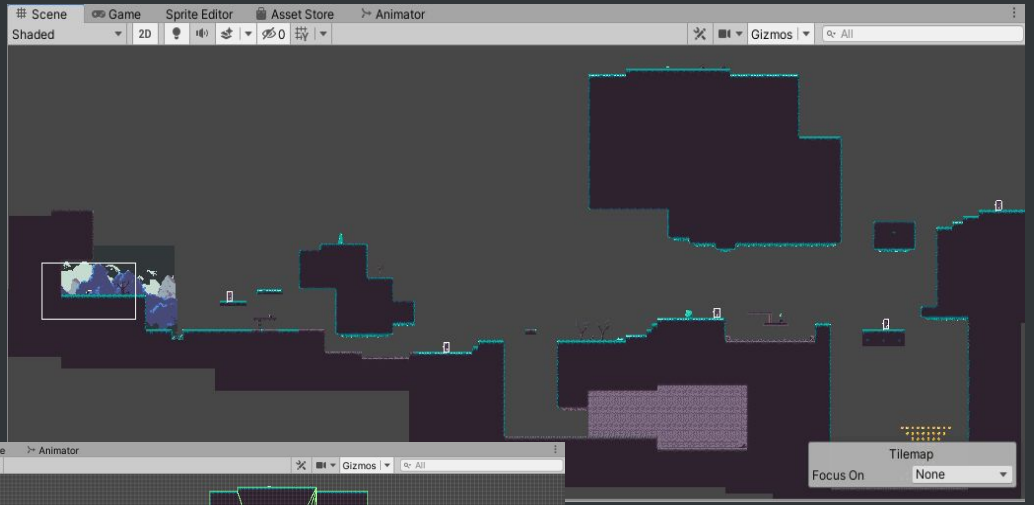


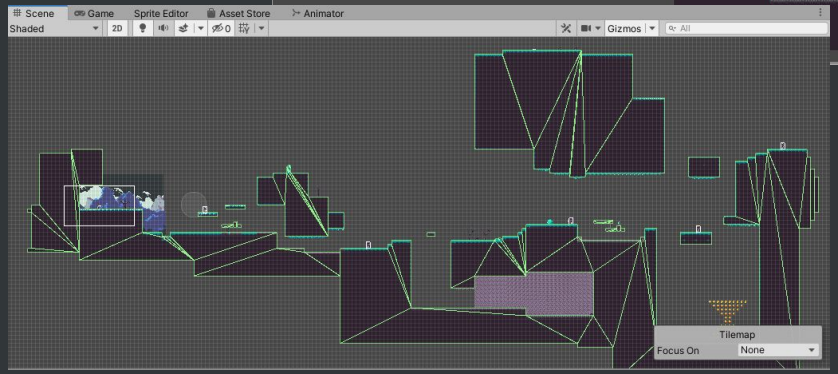
Walk



Attacking

Level Design





Gameplay

Challenges & learning experiences

- Working on this project solo
- Using Unity in a 2D environment rather than 3D
- Code not working properly with character sprites and animations.
- Animation states facing opposite directions
- Clipping through walls
- Time constraints
- Learning how to rig up animations to character sprites
- Different States and state data for multiple use cases
- Parallax backgrounds
- Level design is difficult but rewarding
- Frequent meetings with advisor

Future additions

- Different enemies
- Story and Dialogue
- Multiple weapons
- Adding more levels
- Environmental dangers
- More Sfx
- Collectibles

Thank You!