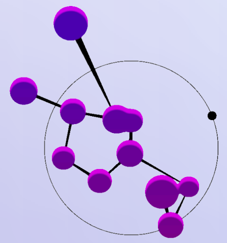
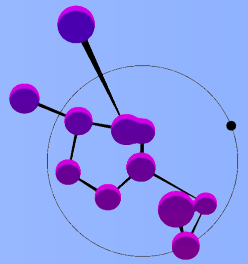
GROUP 7

FINAL GAME PROJECT G7

github: group7

PROJECT OVERVIEW





The Final Project Game is a **2D platformer** created using **MonoGame**, focused on precise movement, platform navigation, and object interactions

core programming concepts:

- Player *movement* and *control* logic
- *Collision* detection
- *Scoring* and *collectibles*
- Platforming mechanics

GROUP 7

KEY CONCEPTS

GAME1.CS - MAIN GAME LOGIC

Game1.cs handles the overall game flow, updating objects and rendering graphics.

Key components include:

1. **Initialize()**: Configures initial game settings.

2. **LoadContent()**: Loads textures, sprites, and other resources.

3. **Update()**: Processes user input, updates game objects, and checks for interactions.

4. **Draw()**: Renders game elements onto the screen.

KEY CONCEPTS

GAME1.CS - MAIN GAME TERMS USED

1. **Rectangle**
   - Definition:
     - A structure in MonoGame (or C#) that represents a rectangle, defined by its position (X, Y), width, and height.
   - Usage:
     - Used for defining the bounding area of game objects (e.g., players, enemies, platforms) for collision detection.

2. **Vector2**
   - Definition:
     - A structure in MonoGame that represents a 2D vector, typically used for positions, directions, and velocities in games.
   - Usage:
     - Represents coordinates (X, Y) or directional movement in 2D space.

GROUP 7

KEY CONCEPTS

GAME1.CS - MAIN GAME TERMS USED

3. **GamePad**
   - Definition:
     - A MonoGame class for handling input from game controllers (e.g., Xbox, PlayStation).
   - Usage:
     - Used to detect button presses, joystick movements, and triggers from a game controller.

4. **Keyboard.GetState()**
   - Definition:
     - A MonoGame method to check the current state of the keyboard for key presses.
   - Usage:
     - Used to detect user input via the keyboard in real-time.

5. **MathHelper.Clamp()**
   - Definition:
     - A utility function in MonoGame's MathHelper class that restricts a value within a specified range.
   - Usage:
     - Used to ensure that a value (e.g., position, velocity, or angle) does not exceed a minimum or maximum threshold.

PLAYER - MOVEMENT AND INTERACTION

- a Player class with properties for position (Position), movement velocity (Velocity), and a jumping state (IsJumping), and initializes these properties in the constructor, setting the position to the specified starting position, velocity to zero, and jumping state to false.

- [Highlighted Code] block handles left and right movement by adjusting the player's X-coordinate.

Highlighted Code:

```
if (Keyboard.GetState().IsKeyDown(Keys.Left))
       { position.X -= speed; }
else if(Keyboard.GetState().IsKeyDown(Keys.Right))
       { position.X += speed; }
```

PLATFORM

PLATFORM COLLISION

The Platform class represents a platform in the game with a specified rectangular area (Rectangle) for defining its dimensions and position.

This ensures the player lands correctly on a platform and stops downward motion.

Highlighted Code:

```
if (player.Bounds.Intersects(platform.Bounds)
&& player.Velocity.Y > 0)
   {
       player.Velocity.Y = 0;
       player.isGrounded = true;
   }
```

GROUP 7

COIN.CS

COLLECTIBLE ITEMS

The Coin class represents a collectible coin in the game with a position (Position) and a state (Collected) indicating whether it has been collected.
Highlighted Code:

```
if (player.Bounds.Intersects(this.Bounds))
 {
     isVisible = false;
     score += 10;
  }
```

Explanation: When a coin is collected, it disappears from the screen and the player's score increases.

GROUP 7

COLLISION DETECTION

- Collision detection is used to check interactions between the player and other objects.

- [Highlighted Code] This code defines bounding boxes for objects and checks for overlaps to detect collisions

Highlighted Code:

```
Rectangle playerBounds = new
Rectangle((int)player.Position.X, (int)player.Position.Y,
player.Width, player.Height);

Rectangle enemyBounds = new
Rectangle((int)enemy.Position.X, (int)enemy.Position.Y,
enemy.Width, enemy.Height);

if (playerBounds.Intersects(enemyBounds))
   {  gameState = GameState.GameOver; }
```

Some Codes to explain:

Code:

```
 if (_player.Position.Y + _playerTexture.Height > platform.Rectangle.Y &&
    _player.Position.Y < platform.Rectangle.Y &&
    _player.Position.X + _playerTexture.Width > platform.Rectangle.X &&
    _player.Position.X < platform.Rectangle.X + platform.Rectangle.Width)
{
    // Allow collision only if the player is falling (moving downward)
    if (_player.Velocity.Y >= 0) // Falling
    {
        _player.Position.Y = platform.Rectangle.Y - _playerTexture.Height;
        _player.Velocity.Y = 0;
        _player.IsJumping = false;
    }
    else if (_player.Velocity.Y < 0) // Jumping
    {
        if (_player.Position.X + (_playerTexture.Width / 2) > platform.Rectangle.X &&
            _player.Position.X + (_playerTexture.Width / 2) < platform.Rectangle.X + platform.Rectangle.Width)
        {
            _player.Position.Y = platform.Rectangle.Y + platform.Rectangle.Height;
            _player.Velocity.Y = 0;
        }
    }
}
```

Explanation :

- Goal: Detect collisions between the player and the platforms.
- Conditions Checked: The player's position and dimensions are compared to the platform's rectangle to check for overlap.
- Falling: If the player is falling (_player.Velocity.Y >= 0), their Y position is adjusted to rest on top of the platform. Vertical velocity is reset to 0, stopping downward motion.
- The IsJumping flag is reset since the player is now grounded.
- Jumping: If the player is jumping (_player.Velocity.Y < 0), collision is only considered if the player is directly under the solid part of the platform. If there is a collision, the player is repositioned below the platform and their upward velocity is stopped.

Code :

```
foreach (var enemy in _enemies)
{
    if (_isGameFrozen) continue;

    enemy.Position.X += enemy.Speed;

    if (enemy.Position.X < enemy.MovementBounds.X ||
        enemy.Position.X + _enemyTexture.Width > enemy.MovementBounds.X + enemy.MovementBounds.Width)
    {
        enemy.Speed *= -1; // Reverse direction
    }

    if (_player.Position.X < enemy.Position.X + _enemyTexture.Width &&
        _player.Position.X + _playerTexture.Width > enemy.Position.X &&
        _player.Position.Y < enemy.Position.Y + _enemyTexture.Height &&
        _player.Position.Y + _playerTexture.Height > enemy.Position.Y)
    {
        _isGameFrozen = true;
        _player.Velocity = Vector2.Zero;
        _currentGameState = GameState.GameOver;
        break;
    }
}
```

Explanation:

- Movement Logic:

1. Enemies move along the X-axis at their respective speeds.
2. When an enemy reaches its movement bounds, its speed is reversed (enemy.Speed *= -1), making it move in the opposite direction.

- Collision Detection:

1. If the player overlaps with an enemy's bounding box, the game state transitions to GameOver.
2. The game is frozen (_isGameFrozen = true) and the player stops moving.

GROUP INSIGHT

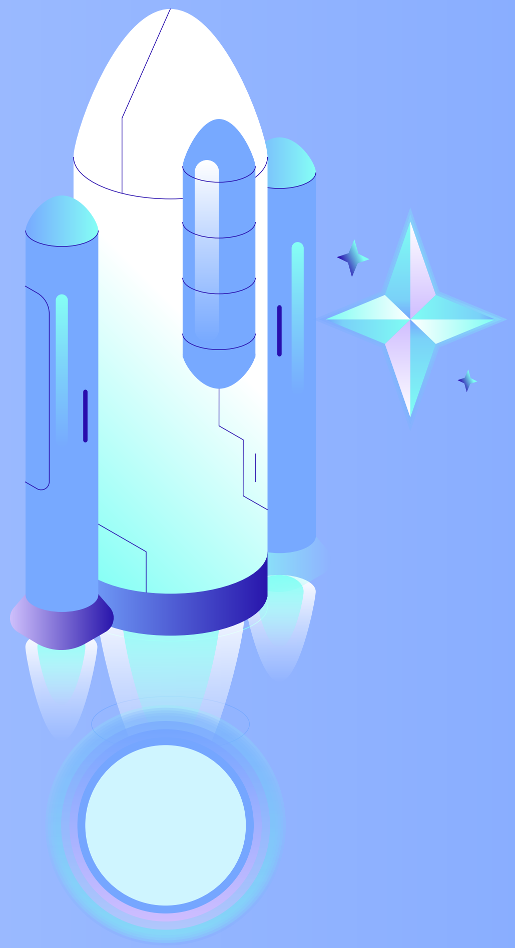
Challenges:

- Ensuring smooth player movement across platforms.
- Implementing precise collision detection.
- Managing the rendering order of objects.

Solutions:

- Used delta-time for consistent movement.
- Employed bounding box logic for collision detection.
- Organized draw calls based on layer priorities.

FUTURE ENHANCEMENTS



The Final Project Game successfully integrates fundamental game development concepts:

- Player movement and interactions
- Collision detection and platforming mechanics
- Scoring system with collectibles

Future Enhancements:

- Add more levels and challenges
- Introduce advanced visual effects and animations
- Expand the scoring system with bonuses and achievements

GROUP 7

THANK YOU!

Team Member:

1. Thi Binh Duong Nguyen _ 8947917
2. Nirav Saxena _ 8979968
3. Aryan Gajjar _ 8972211
4. Ayush Patel _ 8962320