Unit 3 Lab

Player Control

Steps:

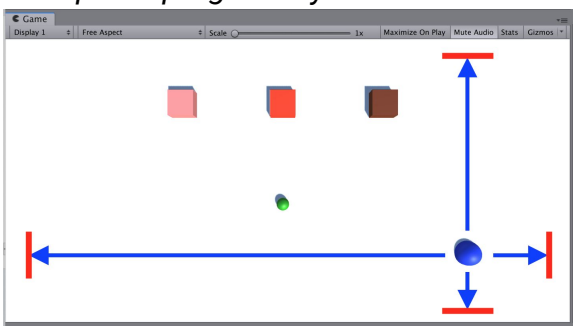
Step 1: Create PlayerController and plan your code

Step 2: Basic movement from user input

Step 3: Constrain the Player's movement

Step 4: Code Cleanup and Export Backup

Example of progress by end of lab



Length: 60 minutes

Overview: In this lesson, you program the player's basic movement, including the code that limits that movement. Since there are a lot of different ways a player can move, depending on the type of project you're working on, you will not be given step-by-step instructions on how to do it. In order to do this, you will need to do research, reference other code, and problem-solve when things go wrong.

Project Outcome: The player will be able to move around based on user input, but *not* be able to move where they shouldn't.

Learning Objectives: By the end of this lab, you will be able to:

- Program the type of player movement you want based on user input
- Restrict player movement in the manner that is appropriate, depending on the needs of the project
- Troubleshoot issues and find workarounds related to player movement

Step 1: Create PlayerController and plan your code

Regardless of what type of movement your player has, it'll definitely need a PlayerController script

1. Select your Player and add a **Rigidbody** component (with or without gravity enabled)
2. In your Assets folder, create a new "Scripts" folder
3. Inside the new "Scripts" folder, create a new "PlayerController" C# script
4. **Attach** it to the player, then **open** it
5. Determine what type of programming will be required for your Player

- **Tip:** Rigidbody is usually helpful - also detect triggers
- **Tip:** Think about all the movement we've done so far:
  - Prototype 1 - forward/back and rotate based on up/down and left/right arrows
  - Challenge 1 - plane moving constantly, rotated direction based on arrows
  - Prototype 2 - side-to-side movement and spacebar to fire a projectile
  - Challenge 2 - No player movement, but projectile launch on spacebar
  - Prototype 3 - background move, and player jumps on spacebar press
  - Challenge 3 - background move and player floats up when spacebar down
- **Don't worry:** If you want your player to move like the ball in Prototype 4, just use basic alternative for now

References to the various types of movement programmed up to this point in the course

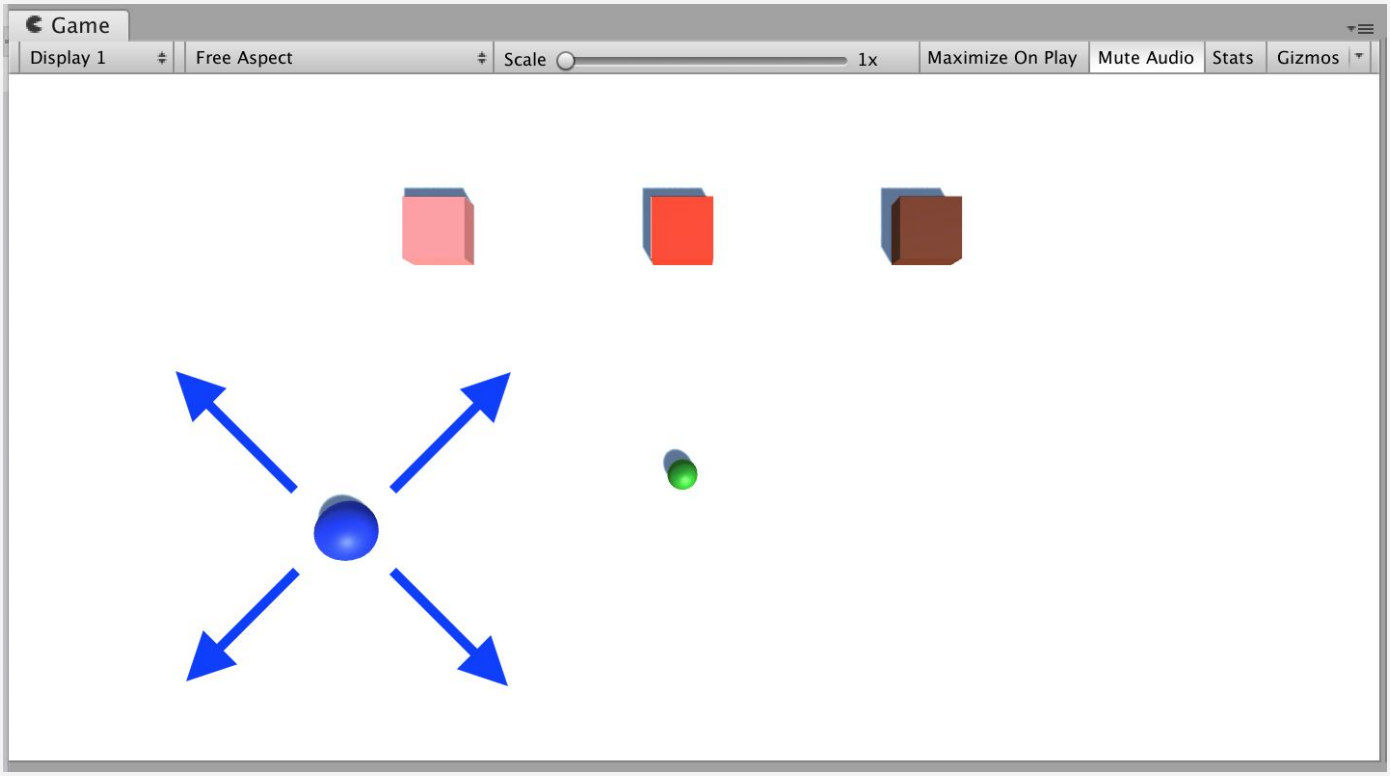
By the end of this step, you should have a new Script open and a solid plan for what will go in it.

Step 2: Basic movement from user input

The first thing we'll program is the player's very basic movement based on user input

1. Declare a new ***private float speed*** variable
2. If using physics, declare a new ***Rigidbody playerRb variable*** for it and initialize it in Start()
3. If using arrow keys, declare new ***verticalInput*** and/or ***horizontalInput*** variables
4. If basing your movement off a key press, create the **if-statement** to test for the **KeyCode**
5. Use either the ***Translate*** method or ***AddForce*** method (if using physics) to move your character

- **Explanation:** Rigidbody movement with AddForce is different than Translate - looks more similar to real world movement with force being applied
- **Don't worry:** If your player is colliding with the ground or other objects in weird ways - we'll fix that soon
- **Tip:** You can look through your old code for references to how you did things



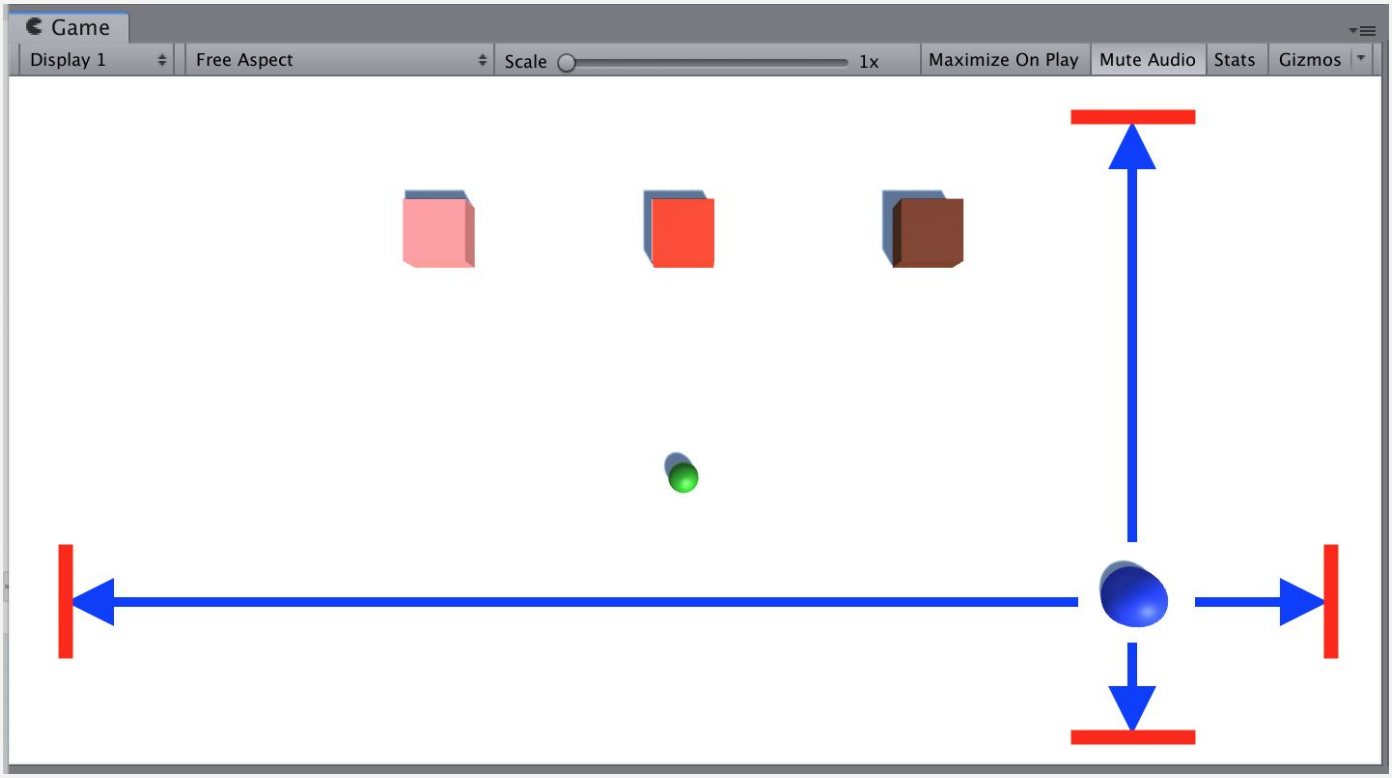
By the end of this step, the player should be able to move the way that you want based on user input.

Step 3: Constrain the Player's movement

No matter what kind of movement your player has, it needs to be limited for gameplay

1. If your player is colliding with objects they shouldn't (including the ground), check the "**Is trigger**" box in the Collider component
2. If your player's position or rotation should be constrained, expand the **constraints** in the Rigidbody component and constrain certain axes
3. If your Player can go **off the screen**, write an **if-statement** checking and resetting the position
4. If the Player can double-jump or fly off-screen, create a **boolean variable** that limits the user's ability to do so
5. If your player should be constrained by physical barriers along the outside of the play area, create more primitive **Planes** or **Cubes** and scale them to form walls

- **Tip:** Check the Global/Local checkbox above scene view to see the rotation of the player
- **Tip:** Look back at Prototype 2 for the if-then statement to keep the player on screen
- **Tip:** Look back at Prototype 3 and Challenge 3 for examples of booleans to prevent double-jumping or going too high



By the end of this step, the player's movement should be constrained in such a way that makes your game playable.

Step 4: Code Cleanup and Export Backup

Now that we have the basic functionality working, let's clean up our code and make a backup.

1. Create new **Empty** game objects and nest objects inside them to **organize** your hierarchy
2. Clean up your Update methods by moving the blocks of code into new void functions (e.g. "MovePlayer()" or "ConstrainPlayerPosition()")
3. Add comments to make your code more readable
4. **Test** to make sure everything still works, then **save** your scene
5. Right-click on your *Assets folder* > ***Export Package*** then save a new version in your **Backups** folder

- **Tip:** You always want to keep your Update() functions clean or they can become overwhelming - it should be easy to see what actions are happening every frame

```
// Move the player left/right and up/down based on arrow keys
void MovePlayer() {
  ...
}

// Prevent the player from leaving the screen top/bottom
void ConstrainPlayerPosition() {
  ...
}
```

By the end of this step, your code should be commented, organized, and backed up.

Lesson Recap

New Progress	• Player can move based on user input • Player movement is constrained to suit the requirements of the game
New Concepts and Skills	• Program in C# independently • Troubleshoot issues independently