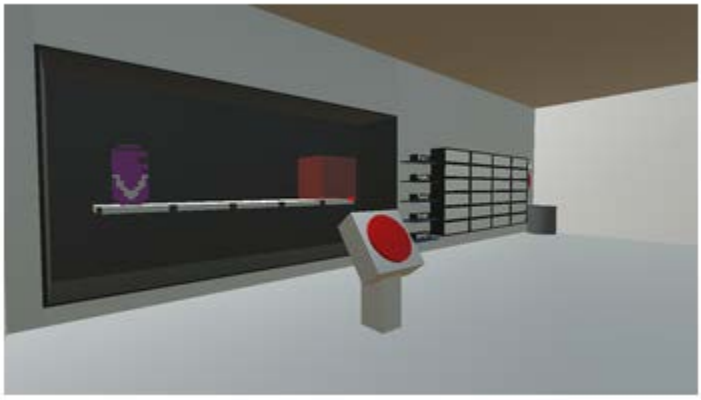
EDUCATIONAL OUTREACH
FLORIDA POLYTECHNIC UNIVERSITY

FLORIDA POLYTECHNIC
LAB CODE ACTIVITY

Introduction to Gaming

Overview:

Congratulations! You've been selected to Beta-Test a new game, created by a new game developer here at Florida Poly! Keegan Sommer graduated from Florida Polytechnic in 2020 with a degree in Computer Engineering. While at Florida Polytechnic he worked in the Office of Educational Outreach where he created this game for students learning how to design video games. Now he needs you to test it for him. What is Beta-Testing? I'm glad you asked! Beta-Testing is an important step in any game's development, where the potential players play the game to make sure the game is both functional, and fun to play.

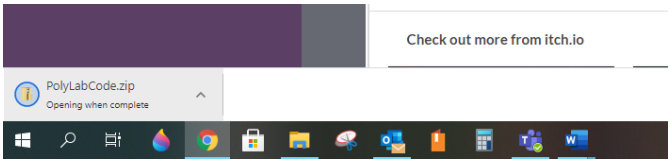


Download

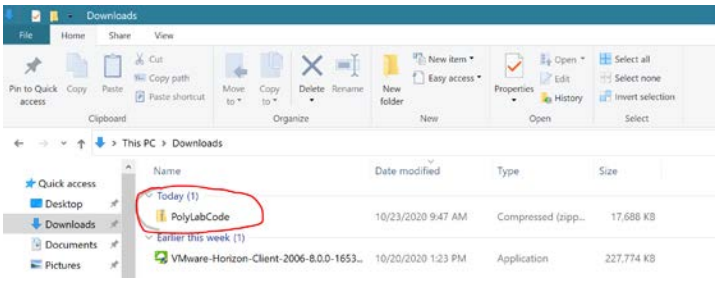
1. To download Poly Lab Code, visit https://keeganjohn.itch.io/florida-polytechnic-lab-code-activity

2. Scroll down and click on the download button

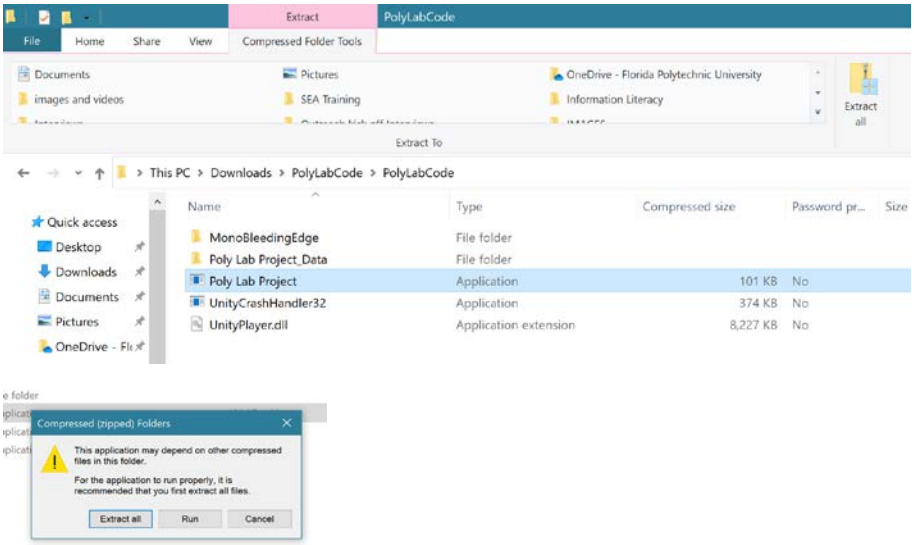
3. A compressed zip folder will download



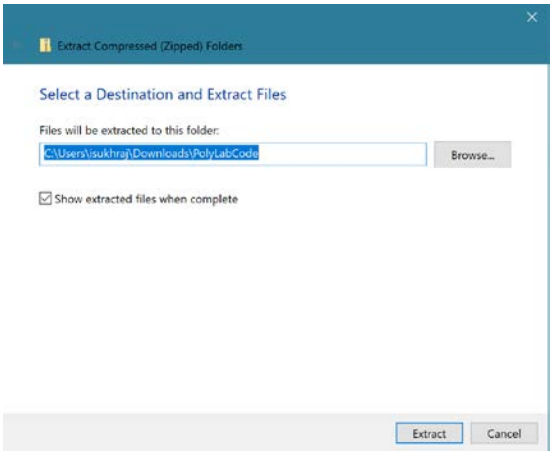
4. Open the zip folder named "PolyLabCode"



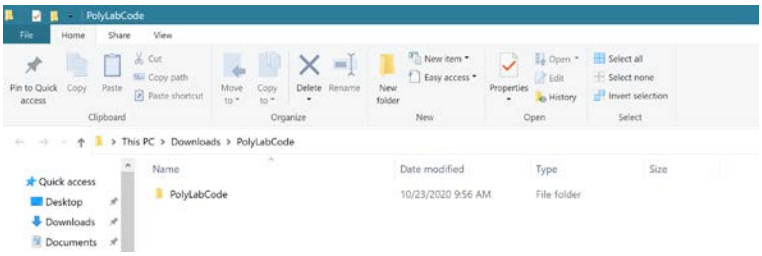
5. Select “Poly Lab Project” and extract all files

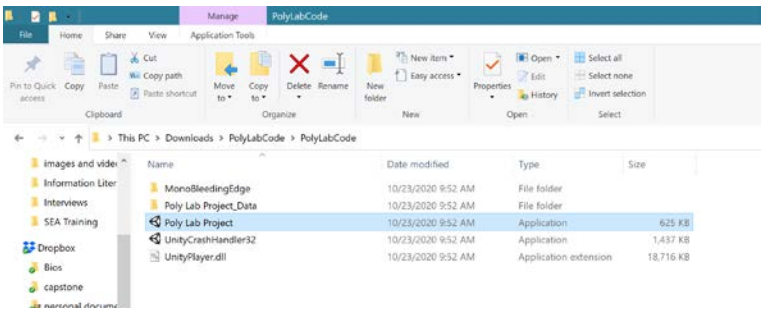


6. Allow your computer to extract files



7. Once the files have been extracted, choose the “Poly Lab Project” in the PolyLabCode folder





8. The Poly Lab Game will open and you are ready to play!

Play The Game

- **Look at the level provided for you to solve and determine what actions the character needs to take to get to the goal.** In this step, you are creating “Pseudocode”. Pseudocode is what programmers create before writing the actual code, to ensure they understand what needs to be done.
- **Assort the provided functions into the order necessary to complete the level.** REMEMBER: Code reads DOWN FIRST. Here, you are using a custom block-based coding language called “Poly Blocks” to write the code that will determine what actions your character will take.
- **Hit the “Run Simulation” button and test your code.** Now after you’ve spent the time to write your program, you get to run the simulation and see if you accomplished your goal. You should watch this step carefully, taking note of where any problems may arise.

- **If the simulation fails, repeat steps 1 to 3.** If and when the simulation fails, don't be discouraged! Failure is a huge part of coding and the creative process involved. All it means is that you made a mistake somewhere, and have to re-work your solution.
- **If the simulation succeeds, congratulations!** You've successfully written a code to navigate the level, and must now begin the process again for the next level.
- **While you are running your simulations, look for any parts of the game that could use improving, or that just don't work**. This is the step that was mentioned earlier called "Beta Testing". If you find anything, be sure to let one of the creators know!

Just for Fun!

The code bellow runs one specific object within the game, the "Test Subject". The Test Subject is the purple guy you see within the simulation room. The "switch" statement you see at the top reads the code that you have given, and tells the test subject to run one of the provided functions (Walk, Run, Jump, Turn, Wait). Then, it runs the appropriate function for the command given.

```
switch (function)
        {
           case "Walk":
              Walk();
              break;
           case "Run":
              Run();
              break;
           case "Jump":
              Jump();
              break;
           case "Turn":
              Turn();
              break;
           case "Wait":
              Wait();
              break;
        }
void Jump()
  {
     if (!jumped)
     {
        jumped = true;
        if(CheckForTile())
           rb.velocity = CalculateJumpForce();
     }
  }
  void Walk()
  {
```

```
    Vector3 vel = rb.velocity.y * transform.up + speed * transform.forward;
    rb.velocity = vel;
  }
  void Run()
  {
    Vector3 vel = rb.velocity.y * transform.up + speed* 2 * transform.forward;
    rb.velocity = vel;
  }
  void Turn()
  {
    if (timeTurning < timeBetweenSteps)
    {
      timeTurning += Time.deltaTime;
    }
    if(timeTurning > timeBetweenSteps)
    {
      timeTurning = timeBetweenSteps;
    }

 transform.localEulerAngles =
Vector3.Lerp(angles,new Vector3(angles.x,angles.y + 180,angles.z), timeTurning / timeBetweenSteps);
    Wait();
  }
  void Wait()
  {
    rb.velocity = new Vector3(0, rb.velocity.y, 0);
  }
```

Created By Keegan Sommer, Computer Engineering ‘20

Thank you for downloading this lesson, please take a moment to complete our survey